

```

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 #include <cstdlib>
27 using namespace std;
28
29
30 /* VARIABLES GLOBALES */
31
32 int HEURE_COURANTE = 0;
33 std::list<event> MES_EVENTS ;
34 std::list<chronique> MES_CHRONIQUES ;
35 std::vector<elmt_sequence> MA_SEQUENCE ;
36
37
38 /* FUNCTIONS */
39
40 chronique check_validation(chronique chro, event e) {
41
42     // mise a jour de chronique dans la liste des chroniques quand on a trouvé le bon event
43     // dans la liste des events (recherche par nom)
44     std::vector<contrainte> list_contraintes;
45     list_contraintes = chro.get_contraintes();
46
47     // indice represente l'indice du tableau de contrainte
48     int indice = (chro.get_contraintes_total()) - (chro.get_contraintes_restantes());
49
50     /* Affectation des conditions générales */
51
52     // TYPE 1/2 : temp sur la condition sur le nom
53     bool cond_nom = ((list_contraintes[indice].get_event().get_label()) == (e.get_label()));
54
55     // TYPE 2 : temp sur la condition sur le temps
56     bool cond_temps;
57
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 // >> valeur du temps limite
71 int contr_temps = list_contraintes[indice].get_time();
72 // >> affectation du booléen
73 cond_temps = (HEURE_COURANTE <= (contr_temps) + last_valid);
74
75 // TYPE 3 : temp sur la condition sur le temps pour le no event (utilise une partie du TYPE 2)
76 // >> ajout de la contrainte de temps et comparaison avec l'heure courante
77 // >> vrai si la contrainte est validée
78 bool cond_temps_no_event = (HEURE_COURANTE >= (last_valid+contr_temps));
79
80 switch(list_contraintes[indice].get_type()){
81
82     case 1: // A pour t(A) quelconque
83
84         if (cond_nom) {
85             // contrainte validée, on passe à la contrainte suivante
86             int nb = chro.get_contraintes_restantes();
87             chro.set_contraintes_restantes(nb -1);
88         }
89         break;
90
91     case 2: // A arrive avant T après le dernier event
92
93         if (cond_nom && cond_temps) {
94
95             // contrainte validée, on passe à la contrainte suivante
96             int nb = chro.get_contraintes_restantes();
97             chro.set_contraintes_restantes(nb-1);
98         }
99         break;
100
101     case 3: // Pas d'évènement A depuis le dernier élément pendant un temps T
102
103         if (cond_temps_no_event) {
104
105             // contrainte validée, on passe à la contrainte suivante
106             int nb = chro.get_contraintes_restantes();
107             chro.set_contraintes_restantes(nb-1);
108
109         } else { // si l'évènement non voulu apparait : remise à zéro de la chronique
110
111             chro.set_contraintes_restantes(chro.get_contraintes_total());
112
113         }
114         break;
115
116     case 4: // A déclenché par une chronique
117
118         // parcours de toute la liste des chroniques
119         for(list<Chronique>::iterator chr_ev=MES_CHRONIQUES.begin(); chr_ev!=MES_CHRONIQUES.end(); chr_ev++) {
120
121             // on regarde si la chronique dont dépend notre chronique
122             // a déjà été validée (chercher par nom et bool à 1)
123             string nom_contr = (list_contraintes[indice].get_event().get_label());
124             bool cond_chr_de_chr = ( nom_contr == ((*chr_ev).get_nom()) ) && ( (*chr_ev).get_event() );
125             if ( cond_chr_de_chr ) {
126
127                 // contrainte validée, on passe à la contrainte suivante
128                 int nb = chro.get_contraintes_restantes();
129                 chro.set_contraintes_restantes(nb-1);
130
131                 // validation de l'évènement associé à la chronique

```

```

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

```

```

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

```

```

263
264
265     }
266
267     }
268
269     }
270
271     /* Affichage de toutes les chroniques à chaque itération */
272     interface();
273
274 }
275
276
277
278 /***** PROCEDURES DE TEST *****/
279
280
281 // TEST DE FONCTIONNEMENT : AFFICHAGE DE LA SEQUENCE LUE
282
283 /*
284 cout<<"La sequence : "<<endl;
285 afficheur_sequence(MA_SEQUENCE);
286
287 */
288
289
290
291 // TEST DE FONCTIONNEMENT : AFFICHAGE DES EVENEMENTS LUS
292
293 /*
294 cout<<"Les evenements : "<<endl;
295 afficheur_liste_evt(MES_EVENTS);
296
297 */
298
299
300 // TEST DE FONCTIONNEMENT : AFFICHAGE DES CHRONIQUES LUES
301
302 /*
303 cout<<"Les chroniques : "<<endl;
304 afficheur_liste_chr(MES_CHRONIQUES);
305 */
306
307
308
309 return 0;
310 }

```