```
/*

--SYSTEME DE RECONNAISSANCE DE CHRONIQUES--

Code developpe pour le projet tuteuré "SURVEILLANCE DES PERSONNES DANS L'HABITAT INTELLIGENT" (2018)
par Valentine Bellet, Benjamin Blum, Amélie Meyer, Ninni Stenkula et Amaury Vignolles

Compilation : g++.exe -Wall -fexceptions -g -std=c++14  -c C:\Users\acer\Desktop\projet-tut-cpp\main.cpp -o obj\Debug\main.o
IDE : Code::Blocks 16.01

Fonctionnement : le fichier .exe doit être à la racine de /src et /txt.

*/

#include <iostream>
#include <string>
#include <vector>
#include <list>
#include "chronique.h"
#include "event.h"
#include "contrainte.h"
#include "sequence.h"
#include "parser.h"
#include "parser_chr.h"
#include "parser_evt.h"
using namespace std;


/* VARIABLES GLOBALES */

int HEURE_COURANTE = 0;
std::list<event> MES_EVENTS ;
std::list<chronique> MES_CHRONIQUES ;
std::vector<elmt_sequence> MA_SEQUENCE ;


/* FUNCTIONS */

chronique check_validation(chronique chro, event e) {

    // mise a jour de chronique dans la liste des chroniques quand on a trouvé le bon event
    // dans la liste des events (recherche par nom)
    std::vector<contrainte> list_contraintes;
    list_contraintes = chro.get_contraintes();

    // indice represente l'indice du tableau de contrainte
    int indice = (chro.get_contraintes_total()) - (chro.get_contraintes_restantes());

    // affectation des conditions
    bool cond_nom = ((list_contraintes[indice].get_event().get_label()) ==  (e.get_label())); // temp sur la condition sur le nom
    bool cond_temps = ((list_contraintes[indice].get_time()) >=  (e.get_last_h_occur())); // temp sur la condition sur le temps
    bool cond_temps_no_event = (HEURE_COURANTE >=  ((e.get_last_h_occur())) + (list_contraintes[indice].get_time())); // temp sur la condition sur le temps pour le no event

    switch(list_contraintes[indice].get_type()){

        case 1: // A pour t(A) quelconque

            if (cond_nom) {

                // contrainte validée, on passe à la contrainte suivante
                int nb = chro.get_contraintes_restantes();
                chro.set_contraintes_restantes(nb -1);
```

```
 63             }
 64             break;
 65 
 66         case 2: // A arrive avant T après le dernier event
 67 
 68             if (cond_nom && cond_temps) {
 69 
 70                 // contrainte validée, on passe à la contrainte suivante
 71                 int nb = chro.get_contraintes_restantes();
 72                 chro.set_contraintes_restantes(nb-1);
 73             }
 74             break;
 75 
 76         case 3: // Pas d'évènement A depuis le dernier élément pendant un temps T
 77 
 78             if (cond_nom && cond_temps_no_event) {
 79 
 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 4: // A déclenché par une chronique
 87 
 88             // parcours de toute la liste des chroniques
 89             for(list<chronique>::iterator chr_ev= MES_CHRONIQUES.begin(); chr_ev!=MES_CHRONIQUES.end();
chr_ev++){
 90 
 91                 // on regarde si la chronique dont dépend notre chronique a déjà été validée (chercher par
nom et bool à 1)
 92                 bool cond_chr_de_chr = ( (list_contraintes[indice].get_event().get_label()) ==
((*chr_ev).get_nom()) ) && ( (*chr_ev).get_event() );
 93                 if ( cond_chr_de_chr ) {
 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                 }
100 
101             }
102             break;
103 
104         default:
105             cout<<"Erreur : type de contrainte inconnu"<<endl;
106 
107     }
108 
109     // toutes les contraintes ont été validées
110     if ((chro.get_contraintes_restantes()) == 0 ) {
111 
112         chro.validation_chronique();
113 
114     }
115 
116     return chro;
117 }
118 
119 
120 /* MAIN */
121 
122 int main(){
123 
124     /*** Initialisation ***/
125 
```

```
126      // recuperation des évenements
127      MES_EVENTS = parser_evenements();
128
129      // recuperation des chroniques
130      MES_CHRONIQUES = parser_chroniques();
131
132      // recuperation de la sequence
133      MA_SEQUENCE = parser();
134
135      /*** Algorithme de lecture de la sequence ***/
136
137      for(vector<elmt_sequence>::iterator vec= MA_SEQUENCE.begin(); vec!=MA_SEQUENCE.end();vec++) {
138
139          // mise a jour de l'heure courante
140          HEURE_COURANTE = (*vec).get_date();
141
142          // mise a jour de la liste d'event
143          for(list<event>::iterator eve=MES_EVENTS.begin(); eve!=MES_EVENTS.end(); eve++) {
144
145              // mise a jour de l'evenement dans la liste des events quand on a trouvé le bon event
146              // dans la liste des events (recherche par nom)
147              if ( ((*eve).get_label()) == ((*vec).get_label()) ) {
148                  (*eve).set_occured(true);
149                  (*eve).set_h_event((*vec).get_date());
150                  (*eve).set_occurence((*eve).get_nb_occurence()+1);
151
152                  // mise a jour de la liste des chroniques
153                  for(list<chronique>::iterator chr= MES_CHRONIQUES.begin(); chr!=MES_CHRONIQUES.end();
chr++){
154
155                     (*chr)=check_validation((*chr),(*eve));
156
157
158                  }
159
160              }
161
162          }
163
164      }
165
166
167
168      /***************** PROCEDURES DE TEST ******************/
169
170
171      // TEST DE FONCTIONNEMENT  : AFFICHAGE DE LA SEQUENCE LUE
172
173      /*
174      cout<<"La sequence : "<<endl;
175      afficheur_sequence(MA_SEQUENCE);
176
177      */
178
179
180
181      // TEST DE FONCTIONNEMENT  : AFFICHAGE DES EVENEMENTS LUS
182
183      /*
184      cout<<"Les evenements : "<<endl;
185      afficheur_liste_evt(MES_EVENTS);
186
187      */
188
189
190      // TEST DE FONCTIONNEMENT  : AFFICHAGE DES CHRONIQUES LUES
```

```
191
192     /*
193     cout<<"Les chroniques : "<<endl;
194     afficheur_liste_chr(MES_CHRONIQUES);
195     */
196
197
198
199     return 0;
200 }
```