```
/*

--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
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

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

    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);
            }
            break;

        case 2: // A arrive avant T après le dernier event

            if (cond_nom && cond_temps) {

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

        case 3: // Pas d'évènement A depuis le dernier élément pendant un temps T

            if (cond_nom && cond_temps_no_event) {

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

        case 4: // A déclenché par une chronique

            // parcours de toute la liste des chroniques
          for(list<chronique>::iterator chr_ev=MES_CHRONIQUES.begin();chr_ev!=MES_CHRONIQUES.end();chr_ev
++){

                // on regarde si la chronique dont dépend notre chronique
                // a déjà été validée (chercher par nom et bool à 1)
                string nom_contr = (list_contraintes[indice].get_event().get_label());
                bool cond_chr_de_chr = ( nom_contr == ((*chr_ev).get_nom()) ) && ( (*chr_ev).get_event() );
                if ( cond_chr_de_chr ) {

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

                }

          }
            break;

        default:
            cout<<"Erreur : type de contrainte inconnu"<<endl;

    }

    // toutes les contraintes ont été validées
    if ((chro.get_contraintes_restantes()) == 0 ) {

        chro.validation_chronique();

    }

    return chro;
}


/* MAIN */

int main(){

    /*** Initialisation ***/
```

```
132
133     // recuperation des évenements
134     MES_EVENTS = parser_evenements();
135
136     // recuperation des chroniques
137     MES_CHRONIQUES = parser_chroniques();
138
139     // recuperation de la sequence
140     MA_SEQUENCE = parser();
141
142     /*** Algorithme de lecture de la sequence ***/
143
144     for(vector<elmt_sequence>::iterator vec= MA_SEQUENCE.begin(); vec!=MA_SEQUENCE.end();vec++) {
145
146         // mise a jour de l'heure courante
147         HEURE_COURANTE = (*vec).get_date();
148
149         // mise a jour de la liste d'event
150         for(list<event>::iterator eve=MES_EVENTS.begin(); eve!=MES_EVENTS.end(); eve++) {
151
152             // mise a jour de l'evenement dans la liste des events quand on a trouvé le bon event
153             // dans la liste des events (recherche par nom)
154             if ( ((*eve).get_label()) == ((*vec).get_label()) ) {
155                 (*eve).set_occured(true);
156                 (*eve).set_h_event((*vec).get_date());
157                 (*eve).set_occurence((*eve).get_nb_occurence()+1);
158
159                 // mise a jour de la liste des chroniques
160                 for(list<chronique>::iterator chr=MES_CHRONIQUES.begin();chr!=MES_CHRONIQUES.end();chr++){
161
162                    (*chr)=check_validation((*chr),(*eve));
163
164
165                 }
166
167             }
168
169         }
170
171     }
172
173
174
175     /***************** PROCEDURES DE TEST ******************/
176
177
178     // TEST DE FONCTIONNEMENT  : AFFICHAGE DE LA SEQUENCE LUE
179
180     /*
181     cout<<"La sequence : "<<endl;
182     afficheur_sequence(MA_SEQUENCE);
183
184     */
185
186
187
188     // TEST DE FONCTIONNEMENT  : AFFICHAGE DES EVENEMENTS LUS
189
190     /*
191     cout<<"Les evenements : "<<endl;
192     afficheur_liste_evt(MES_EVENTS);
193
194     */
195
196
197     // TEST DE FONCTIONNEMENT  : AFFICHAGE DES CHRONIQUES LUES
```

```
198
199     /*
200     cout<<"Les chroniques : "<<endl;
201     afficheur_liste_chr(MES_CHRONIQUES);
202     */
203
204
205
206     return 0;
207 }
```