```cpp
/*

--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 "sequence.h"
#include "parser.h"
#include "parser_chr.h"
#include "parser_evt.h"
#include "interface.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 générales */

        // TYPE 1/2 : temp sur la condition sur le nom
    bool cond_nom = ((list_contraintes[indice].get_event().get_label()) == (e.get_label()));

        // TYPE 2 : temp sur la condition sur le temps
    bool cond_temps = ((list_contraintes[indice].get_time()) >=  (e.get_last_h_occur()));

        // TYPE 3 : temp sur la condition sur le temps pour le no event
        // >> valeur de la dernière validation de la contrainte précédente
    int last_valid = 0; // dans le cas où il n'y a pas de contrainte précédente
    if (indice >= 1) {
        string nom_last = list_contraintes[indice-1].get_event().get_label();
        for(list<event>::iterator last_ev=MES_EVENTS.begin();last_ev!=MES_EVENTS.end();last_ev++){
            if ( ((*last_ev).get_label()) == nom_last) {
                if ((*last_ev).get_occured()) { // sécurité : ne pas faire d'appel sur un pointeur null
                    last_valid = (*last_ev).get_last_h_occur();
                }
```

```
            }
        }
    }
    int contr_temps = list_contraintes[indice].get_time();
    // >> ajout de la contrainte de temps et comparaison avec l'heure courante
    // >> vrai si la contrainte est validée
    bool cond_temps_no_event = (HEURE_COURANTE >= (last_valid+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_temps_no_event) {

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

            } else { // si l'événement non voulu apparait : remise à zéro de la chronique

                chro.set_contraintes_restantes(chro.get_contraintes_total());

            }
            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);

                    // validation de l'événement associé à la chronique
                    for(list<event>::iterator eve=MES_EVENTS.begin(); eve!=MES_EVENTS.end(); eve++) {

                        // mise a jour de l'evenement dans la liste des events quand on a trouvé le bon
event
                        // dans la liste des events (recherche par nom)
```

```
                        bool condition = (((*eve).get_label()) == (chro.get_nom()));
                        if ( condition ) {
                            (*eve).set_occured(true);
                            (*eve).set_h_event(HEURE_COURANTE);
                            (*eve).set_occurence((*eve).get_nb_occurence()+1);

                        }

                    }
                }
            }
            break;

        case 5: // A n'apparait jamais

            bool cond_never; // condition : l'événement n'a jamais eu lieu

            // recherche de l'événement dans la liste des événements pour savoir s'il a eu lieu
            for(list<event>::iterator ev=MES_EVENTS.begin();ev!=MES_EVENTS.end();ev++){
                if ( ((*ev).get_label()) == list_contraintes[indice].get_event().get_label()) {
                    cond_never = !((*ev).get_occured());
                }
            }

            if (cond_never) {

                // 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();

        // gestions des chroniques de chroniques :
        // remise à zero de event_verify (pour les chroniques de présence)
        if (chro.get_nom()=="PresenceSDB_Personne") {
            std::list<chronique>::iterator it;
            for(it=MES_CHRONIQUES.begin();it!=MES_CHRONIQUES.end();it++){
                if (((*it).get_nom())=="PresenceSalon_Personne"){
                    (*it).set_event(false);
                }
                if (((*it).get_nom())=="PresenceChb_Personne"){
                    (*it).set_event(false);
                }
            }
        }
        if (chro.get_nom()=="PresenceSalon_Personne") {
            std::list<chronique>::iterator it;
            for(it=MES_CHRONIQUES.begin();it!=MES_CHRONIQUES.end();it++){
                if (((*it).get_nom())=="PresenceSDB_Personne"){
                    (*it).set_event(false);
                }
                if (((*it).get_nom())=="PresenceChb_Personne"){
                    (*it).set_event(false);
                }
```

```
            }
        }
        if (chro.get_nom()=="PresenceChb_Personne") {
            std::list<chronique>::iterator it;
            for(it=MES_CHRONIQUES.begin();it!=MES_CHRONIQUES.end();it++){
                if (((*it).get_nom())=="PresenceSalon_Personne"){
                    (*it).set_event(false);
                }
                if (((*it).get_nom())=="PresenceSDB_Personne"){
                    (*it).set_event(false);
                }
            }
        }

    }

    return chro;
}


/* MAIN */

int main(){

    /*** Initialisation ***/

    // recuperation des évenements
    MES_EVENTS = parser_evenements();

    // recuperation des chroniques
    MES_CHRONIQUES = parser_chroniques();

    // recuperation de la sequence
    MA_SEQUENCE = parser();


    /*** Algorithme de lecture de la sequence ***/

    for(vector<elmt_sequence>::iterator vec=MA_SEQUENCE.begin(); vec!=MA_SEQUENCE.end();vec++) {

        // mise a jour de l'heure courante
        HEURE_COURANTE = (*vec).get_date();

        // mise a jour de la liste d'event
        list<event>::iterator eve;
        for(eve=MES_EVENTS.begin(); eve!=MES_EVENTS.end(); eve++) {

            // mise a jour de l'evenement dans la liste des events quand on a trouvé le bon event
            // dans la liste des events (recherche par nom)
            bool condition = (((*eve).get_label()) == ((*vec).get_label()));
            if ( condition ) {
                (*eve).set_occured(true);
                (*eve).set_h_event((*vec).get_date());
                (*eve).set_occurence((*eve).get_nb_occurence()+1);

                // mise a jour de la liste des chroniques
                for(list<chronique>::iterator chr=MES_CHRONIQUES.begin();chr!=MES_CHRONIQUES.end();chr++){

                   (*chr)=check_validation((*chr),(*eve));



                }

            }

```

```
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 }
```