```
#if !defined(CINEMA)
#define CINEMA


#include <string>
#include <iostream>
#include "time.h"

using namespace TIME;

class Film;
class Programmation;
class Planning;
class Salle;
class Cinema;
class Cinematheque;
class CinemaException;


std::ostream& operator<<(std::ostream&, const Film&);
std::ostream& operator<<(std::ostream&, const Programmation&);
std::ostream& operator<<(std::ostream&, const Planning&);
std::ostream& operator<<(std::ostream&, const Salle&);
std::ostream& operator<<(std::ostream&, const Cinema&);

const int NB_MAX_PROG=100;

class CinemaException {
public:
      CinemaException(const std::string& m):info(m){}
      const std::string& GetInfo() const { return info; }
private:
      std::string info;
};




class Film{
public:
      Film(const std::string& t, int d):titre(t),duree(d) {}
      void Afficher(std::ostream& f=std::cout) const;
      const std::string& GetTitre() const { return titre; }
      Duree GetDuree() const { return duree; }
private:
      std::string titre;
      Duree duree;
};

class Cinematheque {
public:
      static Cinematheque& GetInstance();
      static void LibereInstance();
      void AjouterFilm(const std::string& t, int d);
      const Film& GetFilm(const std::string& t) const;
private:
      Cinematheque(unsigned int n):nb_films(0),nb_max(n),tab(new Film*[n]){} // ‡ mettre dans la partie
            privÈe pour en interdire l'utilisation
      ~Cinematheque(); // ‡ mettre dans la partie privÈe pour en interdire l'utilisation
      Cinematheque(const Cinematheque&); // non dÈfini mais ‡ mettre dans la partie privÈe pour en
            interdire l'utilisation
      void operator=(const Cinematheque&); // non dÈfini mais ‡ mettre dans la partie privÈe pour en
            interdire l'utilisation
      // attributs
      Film** tab;
      unsigned int nb_films;
      unsigned int nb_max;
      static Cinematheque* instance; // pointeur sur la seule instance de la classe
};

class Programmation {
private:
      const Film* film;
      Intervalle inter;
public:
      Programmation():film(0),inter(Date(),Date()){}
      Programmation(const Film& f, const Date& debut, const Date& fin):film(&f),inter(debut,fin){}
```

```
	void Afficher(std::ostream& f=std::cout) const;
	Intervalle GetReservation() const { return inter; }
};

class Planning {
	Programmation* progs;
	int nbProgs;
	int taille;
public :
	explicit Planning(int t):progs(new Programmation[t]),taille(t),nbProgs(0){}
	~Planning() { delete[] progs; }
	Planning(const Planning& p);
	Planning& operator=(const Planning& p);
	bool operator<<(const Programmation& p);
	void Afficher(std::ostream& f=std::cout) const;
	int GetNbProgrammations() const  { return nbProgs;  }
	int GetTaille() const  { return taille;  }

	class iterateur {
	public:
		const Programmation& operator*() const { return *prog; }
		bool operator!=(const iterateur& it) { return prog!=it.prog; }
		void operator++() { ++prog; }
		iterateur(const Programmation* p=0):prog(p){}
	private:
		const Programmation* prog;
	};
	iterateur begin() const { return iterateur(progs); }
	iterateur end() const { return iterateur(progs+nbProgs); }
};

class Salle{
private:
	int num;
	Planning planning;
public:
	explicit Salle(int n):num(n),planning(NB_MAX_PROG){}
	int GetNumero() const { return num; }
	Planning& GetPlanning() { return planning; }
	const Planning& GetPlanning() const { return planning; }
	void Afficher(std::ostream& f=std::cout) const { f<<"SALLE Numero "<<num<<"\n"; planning.Afficher
		(f); }
};

class Cinema {
private:
	std::string nom;
	Salle** salles;
	unsigned int nbSalles;
	Cinema(const Cinema& c):salles(0),nbSalles(0){}
	void operator=(const Cinema& c){}
public:
	Cinema(const std::string n, int nb);
	~Cinema();
	Salle& operator[](unsigned int i) { if (i<1||i>nbSalles) throw CinemaException("erreur : cette
		salle n'existe pas"); return *salles[i-1]; }
	const Salle& operator[](unsigned int i) const { if (i<1||i>nbSalles) throw CinemaException("erreur
		: cette salle n'existe pas"); return *salles[i]; }
	void Afficher(std::ostream& f=std::cout) const { f<<"Cinema "<<nom<<"\n"; for(unsigned int i=0; i<
		nbSalles; i++) salles[i]->Afficher(f); }
};



#endif
```

```
#include "cinema.h"


std::ostream& operator<<(std::ostream& f, const Film& x){ x.Afficher(f); return f;}
std::ostream& operator<<(std::ostream& f, const Programmation& x){ x.Afficher(f); return f;}
std::ostream& operator<<(std::ostream& f, const Planning& x){ x.Afficher(f); return f;}
std::ostream& operator<<(std::ostream& f, const Salle& x){ x.Afficher(f); return f;}
std::ostream& operator<<(std::ostream& f, const Cinema& x){ x.Afficher(f); return f;}




void Film::Afficher(std::ostream& f) const {
	f<<titre<<" ("<<duree<<")";
}

void Programmation::Afficher(std::ostream& f) const{
	f<<inter<<" : "<<*film;
}

Planning::Planning(const Planning& p):
	progs(new Programmation[p.taille]),taille(p.taille),nbProgs(p.nbProgs){
	for(int i=0; i<nbProgs; i++) progs[i]=p.progs[i];
}
Planning& Planning::operator=(const Planning& p){
	if (this!=&p){
		delete[] progs;
		taille=p.taille;
		progs=new Programmation [taille];
		for(int i=0; i<nbProgs; i++) progs[i]=p.progs[i];
	}
	return *this;
}

bool Planning::operator<<(const Programmation& p){
	if (nbProgs<taille){
		int i=0;
		while(i<nbProgs){
			if (progs[i].GetReservation()&&p.GetReservation()) return false;
			if (progs[i].GetReservation().GetDebut()<p.GetReservation().GetDebut()) i++; else break
				;
		}
		int j=nbProgs;
		while (j>i) {
			progs[j]=progs[j-1];
			j--;
		}
		progs[i]=p;
		nbProgs++;
		return true;
	}
	return false;
}

void Planning::Afficher(std::ostream& f) const{
	for(int i=0; i<nbProgs; i++) f<<"- "<<progs[i]<<"\n";
}

Cinema::Cinema(const std::string n, int nb):
			nom(n),nbSalles(nb),salles(new Salle*[nb]){
	for(unsigned int i=0; i<nbSalles; i++) { salles[i]= new Salle(i+1); }
}

Cinema::~Cinema() {
	for(unsigned int i=0; i<nbSalles; i++) delete salles[i];
	delete[] salles;
}

Cinematheque::~Cinematheque(){
	for(unsigned int i=0; i<nb_films; i++) delete tab[i];
	delete[] tab;
}

Cinematheque* Cinematheque::instance=0; // un membre statique doit Ítre initialisÈ pour Ítre dÈfini

Cinematheque& Cinematheque::GetInstance(){
	if (!instance) instance= new Cinematheque(10);
```

```
	return *instance;
}
void Cinematheque::LibereInstance(){
	if (!instance) delete instance;
}

void Cinematheque::AjouterFilm(const std::string& t, int d){
	if (nb_films==nb_max){
		Film** newtab = new Film* [nb_max+10];
		for(unsigned int i=0; i<nb_films; i++) newtab[i]=tab[i];
		nb_max+=10;
		delete[] tab;
		tab=newtab;
	}
	tab[nb_films]=new Film(t,d);
	nb_films++;
}

const Film& Cinematheque::GetFilm(const std::string& t) const{
	for(unsigned int i=0; i<nb_films; i++)
		if (tab[i]->GetTitre()==t) return *tab[i];
	// si on arrive l‡, le titre n'est pas bon
	throw CinemaException("Demande d'un film inexistant dans la cinematheque");
}
```

```
#if !defined(CTIME)
#define CTIME

#include<iostream>
#include<iomanip>

namespace TIME {
      /*! \class TimeException
     \brief Classe permettant de g@rer les exceptions des classes du namespace TIME
     */
      class TimeException{
      public:
            //! Constructeur @ partir d'une string
            TimeException(const std::string& m):info(m){}
            const std::string& GetInfo() const { return info; } //<! Retourne l'information stock@e dans
                  la classe
      private:
            std::string info;
      };

      /*! \class Date
     \brief Classe permettant de manipuler des dates standards
     L'utilisation de cette classe n@cessite des dates valides au sens commun du terme.
     D@clenchement d'exception dans le cas contraire
     */
      class Date {
      public:
            //! Constructeur @ partir d'un jour, mois, ann@e
            /*! \param j jour avec 1<=j<=31
          \param m mois avec 1<=m<=12
          \param a ann@e avec a>=0
          */
            Date(short j=1, short m=1, unsigned int a=0):jour(1),mois(1),annee(0){ SetDate(j,m,a); }
            // m@thodes
            unsigned short int  GetJour() const { return jour; } //<! Retourne le jour de la date
            unsigned short int  GetMois() const { return mois; } //<! Retourne le mois de la date
            unsigned int GetAnnee() const { return annee; } //<! Retourne l'ann@e de la date
            void SetDate(unsigned short int j, unsigned short int m, unsigned int a); //!< initialisation
                  de la date
            void SetDateAujourdhui(); //!< initialisation de la date avec la date d'aujourd'hui
            void Afficher(std::ostream& f=std::cout) const; //!< affiche le date sous le format JJ/MM/
                  AAAA
            bool operator==(const Date& d) const; //<! d1==d2 retourne vrai si les deux dates sont @gales
            bool operator<(const Date& d) const; //<! Compare deux dates dans le temps : d1<d2 retourne
                  true si d1 est avant d2
            int operator-(const Date& d) const; //<! Retourne le nombre de jours s@parant les deux dates
            Date Demain() const; //<! Retourne la date du lendemain
            Date operator+(unsigned int nb) const; //<!Retourne la date de dans nb jours
      private:
            // attributs
            unsigned short int jour; // jour entre 1 et 31
            unsigned short int mois; // mois entre 1 et 12
            unsigned int annee;
      };

      /*! \class Duree
     \brief Classe permettant de manipuler des durees
     L'utilisation de cette classe n@cessite des dates valides au sens commun du terme.
     D@clenchement d'exception dans le cas contraire
     */
      class Duree{
      public:
            //! Constructeur @ partir de heure et minute
            /*! \param h heure avec h>=0
          \param m minute avec 0<=m<=59
          */
            Duree(unsigned int h, unsigned int m):nb_minutes(h*60+m) {if (m>59) throw TimeException
                  ("erreur: initialisation duree invalide");}
            //! Constructeur @ partir de minute
            /*! \param m minute avec m>=0
          */
            Duree(unsigned int m):nb_minutes(m) {}
            void SetDuree(unsigned int heures, unsigned int minutes) { if (minutes>59) throw
                  TimeException("erreur: initialisation duree invalide"); nb_minutes=heures*60+minutes; }
            unsigned int GetDureeEnMinutes() const { return nb_minutes; } //<!Retourne la duree en
                  minutes
            double GetDureeEnHeures() const { return double(nb_minutes)/60; } //<!Retourne la duree en
```

```
            heures
      void Afficher(std::ostream& f=std::cout) const { f<<nb_minutes/60<<"H"<<std::setw(2)<<
            nb_minutes%60; } //<!Affiche la duree sous le format hhHmm
private:
      unsigned int nb_minutes;
};

/*! \class Horaire
\brief Classe permettant de manipuler des horaires
L'utilisation de cette classe n@cessite des dates valides au sens commun du terme.
D@clenchement d'exception dans le cas contraire
*/
class Horaire{
public:
      //! Constructeur @ partir de heure et minute
      /*! \param h heure avec 0<=h<=23
    \param m minute avec 0<=m<=59
    */
      Horaire(unsigned short int  h, unsigned short int  m):heure(h),minute(m) {if (h>23||m>59)
            throw TimeException("erreur: initialisation horaire invalide");}
      void SetHoraire(unsigned short int h, unsigned short int m) { if (h>23||m>59) throw
            TimeException("erreur: initialisation horaire invalide"); heure=h; minute=m; }
      void Afficher(std::ostream& f=std::cout) const { f<<heure<<"H"<<minute; } //<!Affiche l
            'horaire sous le format hhHmm
      unsigned short int GetHeure() const { return heure; } //<!Retourne l'heure de l'horaire
      unsigned short int GetMinute() const { return minute; } //<!Retourne les minutes de l'horaire
      bool operator<(const Horaire& h) const; //<! h1<h2 retourne true si h1 est avant h2 dans le
            temps
private:
      unsigned short int  heure;
      unsigned short int  minute;
};

/*! \class Periode
\brief Classe permettant de manipuler des periodes exprim@es en jours/mois/ann@es
L'utilisation de cette classe n@cessite des dates valides au sens commun du terme.
D@clenchement d'exception dans le cas contraire
*/
class Periode{
   public :
      //! Constructeur @ partir de jour/mois/ann@e
      /*! \param j nombre de jours avec 0<=j<=364
    \param m nombre de mois avec 0<=m<=11
    \param a nombre d'ann@es
    */
      Periode(unsigned int j, unsigned int m, unsigned int a);
      void Afficher(std::ostream& f=std::cout) const { f<<"{"<<nb_jours<<" jours, "<<nb_mois<<"
            mois, "<<nb_annees<<" ans}"; }
private:
      unsigned int nb_jours;
      unsigned int nb_mois;
      unsigned int nb_annees;
};

/*! \class Intervalle
\brief Classe permettant de manipuler des intervalles de dates
L'utilisation de cette classe n@cessite des dates valides au sens commun du terme.
D@clenchement d'exception dans le cas contraire
*/
class Intervalle{
public:
      //! Constructeur @ partir de deux dates
      /*! \param d date de d@but de l'intervalle
    \param f date de fin de l'intervalle. On doit avoir d<=f
    */
      Intervalle(const Date & d, const Date & f);
      void Afficher(std::ostream& f=std::cout) const; //<! Affiche l'intervalle de dates
      Date GetDebut() const { return debut; } //<! Retourne la date de d@but de l'intervalle
      Date GetFin() const { return fin; } //<! Retourne la date de fin de l'intervalle
      int GetDuree() const { return fin-debut; } //<! Retourne le nombre de jours s'@coulant entre
            le d@but et la fin de l'intervalle
      bool operator&&(const Intervalle & v) const; //<! I1&&I2 Retourne vrai si il y a intersection
            entre I1 et I2
      Intervalle operator + (const Intervalle & i) const; //<! I1+I2 Retourne un intervalle union
            des 2 intervalles I1 et I2 qui se touchent, ie I2.debut est le jour du lendemain de I1.
            fin
private:
```

```
            Date debut;
            Date fin;
      };

}

std::ostream& operator<<(std::ostream&, const TIME::Date&);
std::ostream& operator<<(std::ostream& f, const TIME::Duree & d);
std::ostream& operator<<(std::ostream& f, const TIME::Horaire & h);
std::ostream& operator<<(std::ostream& f, const TIME::Periode & p);
std::ostream& operator<<(std::ostream& f, const TIME::Intervalle & p);

#endif
```

```cpp
#include <iomanip>
#include "time.h"
#include <ctime>

std::ostream& operator<<(std::ostream& f, const TIME::Date& x){ x.Afficher(f); return f;}
std::ostream& operator<<(std::ostream& f, const TIME::Duree & d){ d.Afficher(f); return f; }
std::ostream& operator<<(std::ostream& f, const TIME::Horaire & h){ h.Afficher(f); return f; }
std::ostream& operator<<(std::ostream& f, const TIME::Periode & p){ p.Afficher(f); return f; }


void TIME::Date::SetDate(unsigned short int j, unsigned short int m, unsigned int a){
      // initialisation de la date, renvoie vrai si la date est valide
      if (a<=3000) annee=a; else throw TimeException("erreur: annee invalide");
      if (m>=1&&m<=12) mois=m; else throw TimeException("erreur: mois invalide");
      switch(m){
        case 1: case 3: case 5: case 7: case 8: case 10: case 12: if (j>=1 && j<=31) jour=j; else throw
                 TimeException("erreur: jour invalide"); break;
        case 4: case 6: case 9: case 11: if (j>=1 && j<=30) jour=j; else throw TimeException("erreur:
                 jour invalide"); break;
        case 2: if (j>=1 && (j<=29 || (j==30 && a%4==0))) jour=j; else throw TimeException("erreur: jour
                 invalide"); break;
      }
}

void TIME::Date::SetDateAujourdhui(){
      // initialisation de la date avec la date d'aujourd'hui
      time_t rawtime;
      struct tm * timeinfo;
      time ( &rawtime );
      timeinfo = localtime ( &rawtime );
      jour=timeinfo->tm_mday;
      mois=timeinfo->tm_mon+1;
      annee=timeinfo->tm_year+1900;
}

void TIME::Date::Afficher(std::ostream& f) const{
      // affiche le date sous le format JJ/MM/AAAA
      f<<std::setfill('0')<<std::setw(2)<<jour<<"/"<<std::setw(2)<<mois<<"/"<<annee;
}

bool TIME::Date::operator==(const TIME::Date& d) const{
      if (annee<d.annee) return false;
      if (annee>d.annee) return false;
      if (mois<d.mois) return false;
      if (mois>d.mois) return false;
      if (jour<d.jour) return false;
      if (jour>d.jour) return false;
      return true;
}

bool TIME::Date::operator<(const TIME::Date& d) const{
      if (annee<d.annee) return true;
      if (annee>d.annee) return false;
      if (mois<d.mois) return true;
      if (mois>d.mois) return false;
      if (jour<d.jour) return true;
      if (jour>d.jour) return false;
      return false;
}

int TIME::Date::operator-(const TIME::Date& d) const{
      int n=(annee-d.annee)*365+(annee-d.annee)/4;
      n+=int((mois-d.mois)*30.5);
      n+=jour-d.jour;
      return n;
}

TIME::Date TIME::Date::Demain() const{
      Date d=*this;
      d.jour+=1;
      switch(d.mois){
        case 1: case 3: case 5: case 7: case 8: case 10: case 12: if (d.jour==30) { d.jour=1; d.mois++;
                 } break;
        case 4: case 6: case 9: case 11: if (d.jour==31) { d.jour=1; d.mois++; } break;
        case 2: if (d.jour==29 && d.annee%4>0) { d.jour=1; d.mois++; } if (d.jour==30) { d.jour=1; d.
                 mois++; } break;
      }
```

```
	if (d.mois==13){ d.annee++; d.mois=1; }
	return d;
}

TIME::Date TIME::Date::operator+(unsigned int nb_jours) const{
	Date d=*this;
	while(nb_jours>0) { d=d.Demain(); nb_jours--; }
	return d;
}

bool TIME::Horaire::operator<(const Horaire& h) const{
	if (heure<h.heure) return true;
	if (heure>h.heure) return false;
	if (minute<h.minute) return true;
	if (minute>h.minute) return false;
	return true;
}

TIME::Periode::Periode(unsigned int j, unsigned int m, unsigned int a):
nb_jours(j), nb_mois(m), nb_annees(a) {
	if (j>364) throw TimeException("erreur: initialisation periode invalide");
	if (m>11) throw TimeException("erreur: initialisation periode invalide");
}


std::ostream& operator<<(std::ostream& f, const TIME::Intervalle& x){ x.Afficher(f); return f;}


TIME::Intervalle::Intervalle(const Date & d, const Date & f):debut(d),fin(f){
	if (fin<debut) throw TimeException("Erreur dans la creation d'un intervalle: fin<debut");
}

bool TIME::Intervalle::operator&&(const Intervalle & v) const {
	if (debut<v.debut){
		if (fin<v.debut) return false; else return true;
	}
	if (v.fin<fin){
		if (v.fin<debut) return false; else return true;
	}
	return true;
}

TIME::Intervalle TIME::Intervalle::operator+(const Intervalle & i) const {
	Date d=fin.Demain();
	if (d==i.debut){
		return Intervalle(debut,i.fin);
	}else throw TimeException("Ne peut pas faire l'union de 2 intervalles (ils ne se touchent pas...")
		;
}

void TIME::Intervalle::Afficher(std::ostream& f) const {
	f<<"["<<debut<<" ; "<<fin<<"]";
}
```