Statistiques d'un hôpital [pg02]
Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 21 mai 2018

Table des matières

C++ - Statistiques d'un hôpital (Solution)

Mots-Clés Exercices généraux ■
Requis Axiomatique objet ■
Difficulté ● ● ○ (4 h) ■

Objectif

Cet exercice réalise un programme orienté objet des statistiques d'un hôpital.

1 Problème et Cahier des charges

Problème

Un hôpital veut réaliser :

- Des statistiques sur l'occupation de ses chambres.
- L'affectation des chambres pour les nouveaux patients.

L'hôpital dispose de N chambres et toutes les chambres ont 4 lits. On ajoute un patient dans une chambre si elle n'est pas pleine et qu'elle est parmi les plus occupées de l'hôpital.

Exemple

Voici un hôpital avec $N = 10$ chambres :

1	3	4	0	2	3	4	4	0	1J

Les chambres 3 et 8 sont vides, les chambres 2, 6, 7 sont pleines (les quatre lits sont occupés), les chambres 0 et 9 ont 1 lit occupé, les chambres 1 et 5 ont 3 lits occupés et la chambre 4 a 2 lits occupés.

Si une chambre a 3 lits occupés et une autre en a un, on ajoutera le patient dans la première des deux qui est la plus occupée. Ici on ajoutera donc le premier client dans la deuxième chambre, et le deuxième client sera ajouté dans la sixième.

Cahier des charges

On veut écrire un programme qui :

1. Initialise les chambres de l'hôpital avec les données d'un tableau.
2. Affiche les statistiques :
   - Nombre de chambres pleines
   - Nombre de chambres vides
   - Taux d'occupation global (nombre de patients / nombre de lits)
3. Affecte un patient dans une chambre si c'est possible, et signale une erreur sinon.

2 Solution sans objets

2.1 Les données

On représente les N chambres par un tableau de N cases :

- L'indice représente le numéro de la chambre.
- Le contenu de la case représente le nombre de lits occupés dans cette chambre.

Définissez les constantes :

- `MAXLITS=4` (nombre maximum de lits par chambre).
- `NMAX=20` (nombre maximum de chambres).

Définissez le type `Hopital` comme étant un tableau d'entiers de taille `NMAX`.

2.2 Ajout d'un patient

Ajout d'un patient
Pour ajouter un patient, il faut chercher l'indice de la chambre dont l'occupation est la plus grande parmi celles qui ont toujours de la place. C'est donc une adaptation du programme de recherche de l'indice du maximum d'un tableau. On utilisera deux variables :

- `maxNonPlein` : le maximum de lits occupés pour les chambres non pleines
- `rs` : indice de cette case.

Peut-on initialiser `maxNonPlein` avec le premier élément du tableau ?

Solution simple
Non car cette case peut être pleine et donc avoir le maximum de lits. De même comparer avec elle ne permettra pas de trouver une chambre avec de la place (car elle aura moins de lits).

Comment alors initialiser les deux variables ?

Solution simple
On les initialise à `-1`. A l'issue de la recherche, si `maxNonPlein>=0` alors on a trouvé une chambre. Sinon toutes les chambres sont pleines.

Écrivez une fonction `ajouterPatient(t,n)` qui ajoute un client dans un `Hopital t` de `n` chambres. La fonction renvoie l'indice de la case d'ajout du client, `-1` sinon.

2.3 Statistiques

Écrivez une procédure `calcAffStats(t,n)` qui comptabilise le nombre de lits occupés dans `patients` (entier), de chambres pleines dans `pleines` (entier) et de chambres vides dans `vides` (entier) d'un `Hopital t` de `n` chambres.

Calculez le taux d'occupation des chambres dans `tauxOccup` (réel).

Affichez les statistiques (où `[x]` désigne le contenu de `x`) :

```
Nombre de patients: [patients]
Nombre de chambres pleines: [pleines]
Nombre de chambres vides  : [vides]
Taux d'occupation: [tauxOccup]
```

2.4 Programme principal

Écrivez un programme qui déclare un `Hopital t` et l'initialise avec :

```
t = {1,3,4,0,2,3,4,4,0,1};
int nchambres = 10;
```

Calculez et affichez les statistiques.

Tentez d'ajouter un patient et affichez l'un des deux messages :

```
Hopital plein -- Ajout impossible
Patient mis dans la chambre ...
```

Testez.

Validez vos fonctions, vos procédures et votre programme avec la solution.

Solution C++ @[pghopitalA1.cpp]

```cpp
#include <iostream>
using namespace std;

// Definitions
const int MAXLITS = 4;
const int NMAX = 20;
typedef int Hopital[NMAX];

/**
  Ajoute un patient dans un hopital
  @param[in,out] t - Un Hopital
```

```
  @param[in] n - nombre de chambres
  @return Le numéro de chambre affectee, -1 si l'ajout est impossible
*/
int ajouterPatient(Hopital& t, int n)
{
  int maxNonPlein = -1;
  int rs = -1;
  for (int k = 0; k < n; ++k)
  {
    if (t[k] < MAXLITS and t[k] > maxNonPlein)
    {
      rs = k;
      maxNonPlein = t[k];
    }
  }
  if (maxNonPlein >= 0)
  {
    ++t[rs];
  }
  return rs;
}

/**
  Calcule et affiche les statistiques
  @param[in] t - Un Hopital
  @param[in] n - nombre de chambres
*/
void calcAffStats(const Hopital& t, int n)
{
  int pleines = 0;
  int vides = 0;
  int patients = 0;
  // Calculs
  for (int k = 0; k < n; ++k)
  {
    patients += t[k];
    if (t[k] == MAXLITS)
    {
      ++pleines;
    }
    else if (t[k] == 0)
    {
      ++vides;
    }
  }
  double tauxOccup = 1.0 * patients / (n * MAXLITS);
  // Affichages
  cout<<"Nombre de patients: "<<patients<<endl;
  cout<<"Nombre de chambres pleines: "<<pleines<<endl;
  cout<<"Nombre de chambres vides  : "<<vides<<endl;
  cout<<"Taux d'occupation: "<<tauxOccup<<endl;
}

/**
  Programme de test
*/
int main()
{
```

```
  // Initialisations
  Hopital t={1,3,4,0,2,3,4,4,0,1};
  int nchambres = 10;
  // Statistiques
  calcAffStats(t, nchambres);
  // Ajout d'un patient
  int chAff = ajouterPatient(t,nchambres);
  if (chAff < 0)
  {
    cout<<"Hopital plein -- Ajout impossible"<<endl;
  }
  else
  {
    cout<<"Patient mis dans la chambre "<<chAff<<endl;
  }
}
```

Que pensez-vous de cette solution ?

Solution simple

C'est un programme très rudimentaire :

- Chaque chambre a toujours `MAXLITS` lits.
- On ne considère pas les données des patients.
- Deux uniques fonctionnalités : statistiques et ajout d'un patient.
- Il est impossible d'ajouter un patient dans une chambre déterminée.

3 Solution avec des objets

3.1 Conception OO

Solution objet

Se prête-t-elle à des applications plus sophistiquées ?

- La prise en compte des données de patients
- Des chambres « paramétrées » par le nombre de lits
- Un hôpital « paramétré » par le nombre de chambres et de lits dans chaque chambre
- L'ajout d'un patient dans une chambre donnée
- D'autres opérations (chercher le numéro de chambre d'un patient, sortie d'un patient...)
- Comment savoir quelles classes écrire et quoi mettre dedans ?
- Quelles classes (pour modéliser quoi) ?
- Quelles données et fonctionnalités dans chaque classe ?
- Peut-on écrire « notre » programme principal avec tout ceci ?

Analyse simple

Les programmes objets cherchent à modéliser :

- Des **concepts** ==> modélisés par des classes.
- Des **comportements** ==> modélisés par des méthodes.

Pour les trouver :

- Réécrire la description du problème en soulignant :
- Les noms ==> candidats de concepts (classes).
- Les verbes ==> candidats de comportements (méthodes).

Cahier des charges revu

On veut écrire un programme qui :

1. Initialise les chambres de l'hôpital.
2. Affiche le nombre de chambres pleines de l'hôpital.
3. Affiche le nombre de chambre vides de l'hôpital.
4. Affiche le taux d'occupation de l'hôpital.
5. Ajoute un patient dans une chambre de l'hôpital si c'est possible, et signale une erreur sinon.
6. Supplémentaires :
   - Prendre en compte les données des patients.
   - Ajouter un patient dans une chambre donnée.

3.2 Modélisation OO

Concepts

On en détermine trois :

- `Hopital` : ensemble de chambres avec des lits et des patients dedans
- `Chambre` : ensemble de lits avec ou non des patients dedans
- `Patient` : un nom, un numéro de sécurité sociale, etc.

Comportements

Il y a en deux :

- Afficher les statistiques ==> dans quelle classe ?
- Ajouter un patient ==> dans quelle classe ?

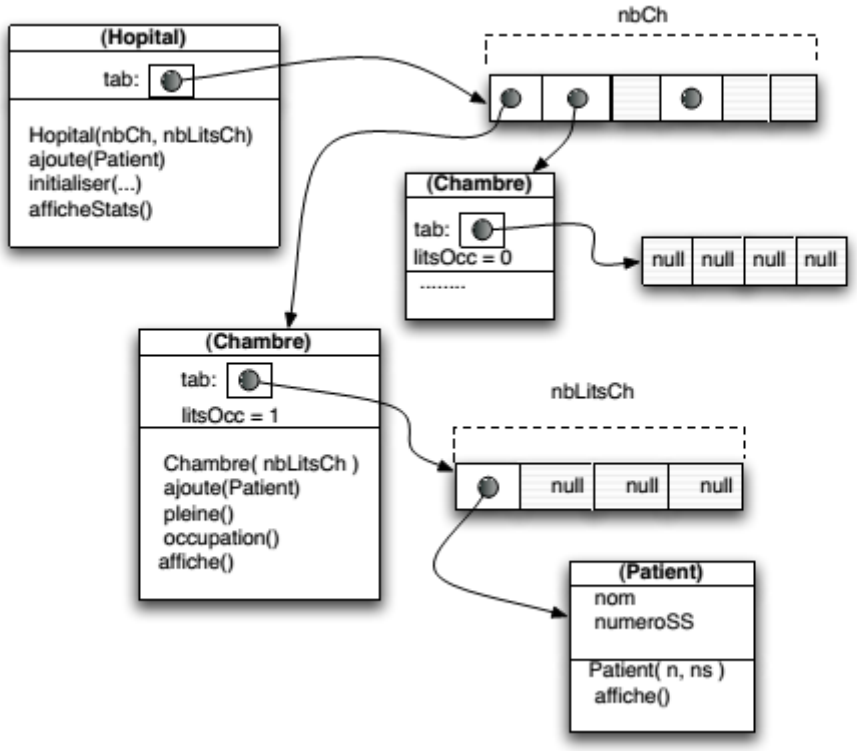
Classes du programme

Quelles classes utilisera-t-on à partir du programme ?

- Classe `Hopital`
  - (variables) ensemble des chambres
  - (méthode) afficher les statistiques
  - (méthode) ajouter un patient
  - (méthode) affichages pour les tests
- Classe `Patient`
  - (variables) données d'un patient
  - (méthode) affichages
- Classe `Chambre`
  - (variables) ensemble de lits, libres ou occupés par un patient
  - (méthodes) affichages et quoi d'autre ?

...(suite page suivante)...

Dessin de la modélisation



3.3 Fonctionnalités et Prototype

Les fonctionnalités données par les méthodes d'une classe sont disponibles pour les autres classes et pour le programme :

==> on doit réfléchir à ce qu'il faut mettre dans chaque classe

Quelles fonctionnalités ?

Pour savoir que mettre dans une classe `A`, on peut se poser les questions suivantes :

1. Qui a besoin de la classe `A` ?
2. Pour quoi faire ?
   ==> donner une liste d'actions.
3. En face de chaque action
   ==> en-tête d'une méthode pour la réaliser.

Cette liste d'en-têtes de méthodes est un point de départ pour établir un **prototype** du contenu de `A`.

Prototype d'une classe

Schéma de ce qui est modélisé par la classe
==> déclaration de variables + en-têtes de méthodes.

Utilité du prototype
Il permet de préciser :

- Comment sont modélisées les données.
- Quelles sont les fonctionnalités fournies par la classe.

En ne donnant que les en-têtes, on ne s'occupe pas des détails d'implantation des méthodes : on réfléchit **à ce qui est nécessaire** au reste du programme **avant de coder**.

3.4 Premier prototype de Patient

Le programme principal et/ou la classe `Hopital` ont besoin de la classe `Patient` pour :

- La création d'un patient.
- Cette création est préalable à l'ajout du patient dans l'hôpital.
- L'affichage des données des patients lors des tests.

Écrivez un premier prototype de la classe `Patient`.
Incluez :

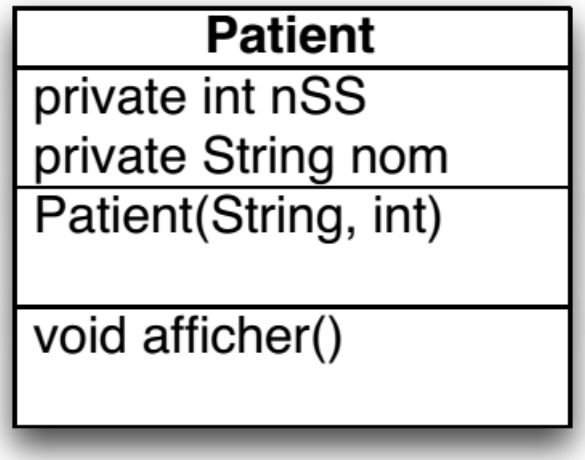
- Son nom `nom` (chaîne de caractères).
- Son numéro de sécurité sociale `nss` (entier).

Fournissez :

- Un constructeur initialisant les attributs.
- Une méthode `afficher`.

Validez votre classe avec son schéma UML.

Solution simple



3.5 Premier prototype de Hopital

Le programme principal a besoin de la classe `Hopital` pour :

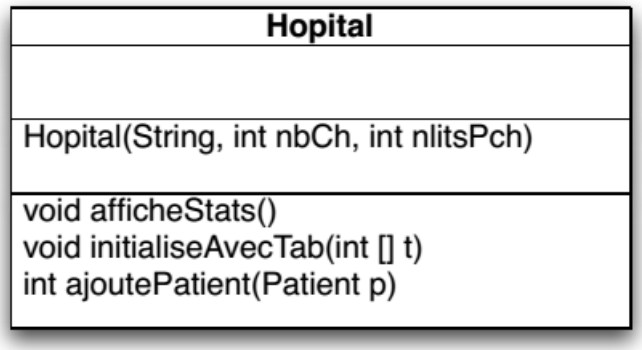
1. La création d'un hôpital
2. L'initialisation de l'hôpital d'après un tableau d'entiers
3. L'affichage des statistiques d'occupation
4. L'ajout d'un patient et affichage de sa chambre d'affectation

Écrivez un premier prototype de la classe `Hopital`.
Fournissez :

- Un constructeur paramétré par son `nom` (chaîne de caractères), le nombre de chambres `nCh` (entier) et le nombre de lits par chambre `nlitsCh` (entier).
- Une méthode `initialiserAvecTableau(t,n)` qui initialise les données d'après un tableau `t` de `n` entiers.
- Une méthode `ajouterPatient(p)` qui ajoute un `Patient p` et renvoie sa chambre d'affectation.
- Une méthode `afficherStats` qui affiche les statistiques d'occupation.

Validez votre classe avec son schéma UML.

Solution simple



3.6 Première version du programme

Une manière de valider ces prototypes est de tenter d'écrire un programme principal. Cela permet de savoir s'il manque des méthodes ou s'il y a d'autres problèmes dans la modélisation.

Écrivez une première version du programme qui initialise un `Hopital h` avec le tableau :

```
t = {1,3,4,0,2,3,4,4,0,1};
int nchambres = 10;
```

Affichez les statistiques.

Ajoutez un `Patient` et affichez le résultat de la tentative.

3.7 Codage de la classe Patient

Écrivez le constructeur par défaut (pour les tableaux).

Écrivez le constructeur normal.

Pour les patients, nous donnons des accesseurs pour tous les attributs. Écrivez les accesseurs `getNom` du nom et `getNSS` du numéro de Sécurité sociale.

Écrivez la méthode `afficher` qui affiche les données du `Patient`.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Patient.hpp] @[Patient.cpp]

```
#ifndef PATIENT_CLASS
#define PATIENT_CLASS

/*
// Premier prototype de Patient
class Patient
{
  public:
    Patient(const string& nom, int nss);
    void afficher() const;
  private:
    string m_nom;
    int m_nss;
};
*/

// Raffinement de la classe Patient
class Patient
{
  public:
    Patient();
    Patient(const string& nom, int nss);
    void afficher() const;
```

```
    string getNom() const;
    int getNSS() const;
  private:
    string m_nom;
    int m_nss;
};
#include "Patient.cpp"
#endif
```

```
// Constructeur par defaut (pour les tableaux)
Patient::Patient()
: m_nom(), m_nss(0)
{}

// Constructeur normal
Patient::Patient(const string& nom, int nss)
: m_nom(nom), m_nss(nss)
{}

// Accesseur du nom
string Patient::getNom() const
{
  return m_nom;
}

// Accesseur du numero de Securite sociale
int Patient::getNSS() const
{
  return m_nss;
}

void Patient::afficher() const
{
  cout<<"  Nom: "<<getNom()<<" -- NumeroSS: "<<getNSS()<<endl;
}
```

3.8 Raffinement de la classe Hopital

Les variables de la classe `Hopital` sont :

- `nom` : nom de l'hôpital
- `t` : `Vecteur` (tableau dynamique) de `Chambre`
- `nbCh` : nombre de chambres (défini dans le vecteur)
- `nLits` : nombre total de lits
- `nLitsOcc` : total de lits occupés
- `nChVides` : total de chambres vides
- `nChPleines` : total de chambres occupées

Les fonctionnalités supplémentaires souhaitées sont :

- Trouver une chambre où ajouter un patient
- Tester si un patient peut être mis dans une chambre donnée
- Ajout d'un patient dans un numéro de chambre donné
- Affichage des statistiques, affichages divers pour les tests

Incluez les attributs dans votre classe.

Fournissez les accesseurs `getNom`, `getNbChambres`, `getNbChPleines`, `getNbChVides` et la méthode `tauxOccup`.

Fournissez les accesseurs `getNom`, `getNbChambres`, `getNbChPleines`, `getNbChVides` et la méthode `tauxOccup` du taux d'occupation.

Ajout d'un patient

L'ajout dans l'hôpital se fait « normalement » dans une des chambres les plus remplies ayant encore de la place, mais cette méthode ne nous permet pas d'initialiser l'hôpital comme dans l'exemple donné (plusieurs chambres avec 1, 2 ou 3 patients). Il convient donc de nouvelles méthodes pour ajouter un patient dans un numéro de chambre donné.

Fournissez :

- Une méthode `chambreDispo(nCh)` qui teste et renvoie `Vrai` si la chambre numéro `nCh` est disponible, `Faux` sinon.
- Une méthode `getChambreDispo()` qui renvoie le numéro d'une chambre disponible.
- Une méthode `ajouterPatient(p,nCh)` qui ajoute et renvoie `Vrai` si un `Patient p| a é té ajouté dans la chambre numéro \lstinlinenCh@`, `Faux` sinon.

Fournissez les méthodes `afficher`, `afficherOccupation`, `afficherPatientsParChambre` et `afficherTout`.

Validez votre prototype avec la solution.

Solution C++

```
// Raffinement de Hopital
class Hopital
{
  public:
    Hopital(const string& nom, int nCh, int nlitsCh);
    void initialiserAvecTableau(const int t[], int n);
    int ajouterPatient(const Patient& p);
    void afficherStats() const;

    string getNom() const;
    int getNbChambres() const;
    int getNbChPleines() const;
    int getNbChVides() const;
    double tauxOccup() const;

    void afficher() const;
    void afficherOccupation() const;
    void afficherPatientsParChambre() const;
```

```
    void afficherTout() const;
  private:
    ....
};
```

3.9 Premier prototype de Chambre

La classe `Chambre| est utilisée par la classe \lstinline`Hopital@ pour :

- La création de chaque chambre de l'hôpital.
- L'ajout d'un patient dans une chambre si cela est possible.
- La détermination de l'occupation d'une chambre, si elle est vide ou pleine, préalable à l'ajout, et nécessaire aux statistiques.
- L'affichage des patients et/ou l'occupation d'une chambre (pour les tests).

Écrivez un premier prototype de la classe `Chambre`.
Fournissez :

- Un constructeur par défaut.
- Un constructeur paramétré par le numéro de la chambre `num` et par sa capacité en lits `nlits`.
- Une méthode `afficher`.
- Une méthode `getOccupation` et des prédicats `estVide` et `estPleine`.
- Une méthode `ajouterPatient(p)` qui ajoute un `Patient p`.

Validez votre prototype avec la solution.

Solution C++

```
// Premier prototype de Chambre
class Chambre
{
  public:
    Chambre();
    Chambre(int num, int nlits);
    int getOccupation() const;
    bool estVide() const;
    bool estPleine() const;
    bool ajouterPatient(const Patient& p);
    void afficher() const;
};
```

3.10 Codage de la classe Hopital

Écrivez le constructeur. Il reçoit en paramètres le nom de l'hôpital, le nombre de chambres et leur capacité en lits.

Aide simple

- Il doit créer un tableau de chambres de la taille appropriée.
- Il ne suffit pas de créer le tableau des N chambres : il faut aussi créer chaque chambre et l'affecter dans les cases du tableau.

Écrivez les accesseurs `getNom` du nom, `getNbChambres` du nombre de chambres, `getNbChPleines` du nombre de chambres pleines et `getNbChVides` du nombre de chambres vides.

Écrivez la méthode `chambreDispo(nCh)` qui teste et renvoie `Vrai` si le numéro de chambre `nCh` dispose d'une place, `Faux` sinon.

Déduisez la méthode `ajouterPatient(p,nCh)` qui ajoute un `Patient p` dans une chambre numéro `nCh`. Elle renvoie `Vrai` si l'opération a été effectuée, `Faux` sinon.

Écrivez la méthode `getChambreDispo` qui trouve la chambre la plus remplie ou renvoie `-1` si toutes sont pleines. Elle utilise le même algorithme que pour la solution sans objets.

Écrivez la méthode `ajouterPatient(p)` qui ajoute un `Patient p`.

Aide simple

Elle utilise la méthode `getChambreDisponible` puis redifère le traitement sur la méthode d'ajout d'un patient dans une chambre donnée.

Écrivez la méthode `initialiserAvecTableau(t,n)` qui initialise l'hôpital avec un tableau d'entiers `t` de taille `n`.

Aide simple

- Elle utilise la méthode qui ajoute un patient dans une chambre donnée.
- Les patients sont crées « automatiquement » avec des noms p1, p2, ... et des numéros 1, 2, 3...

Écrivez la méthode `tauxOccup` qui calcule et renvoie le taux d'occupation.

Écrivez la méthode `afficher` qui affiche :

```
Hopital [nom]
Nb de chambres = [t.taille]
Capacite en lits = [nLits]
```

Écrivez la méthode `afficherStats` qui affiche :

```
Nb lits occupes     = [nLitsOcc]
Nb chambres vides   = [nChVides]
Nb chambres pleines = [nChPleines]
Taux d'occupation   = [tauxOccup]
```

Écrivez la méthode `afficherOccupation` qui affiche le nombre de patients pour chacune des chambres :

```
Chambre [k] : t[k].getOccupation()
```

Écrivez la méthode `afficherPatientsParChambre` qui affiche tous les patients. Elle est utilisée pour les tests.

Enfin écrivez la méthode `afficherTout` qui appelle les quatre méthodes `afficherXXX`.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Hopital.hpp] @[Hopital.cpp]

```cpp
#ifndef HOPITAL_CLASS
#define HOPITAL_CLASS

#include <vector>
using namespace std;

#include "Patient.hpp"
#include "Chambre.hpp"

/*
// Premier prototype de Hopital
class Hopital
{
  public:
    Hopital(const string& nom, int nCh, int nlitsCh);
    void initialiserAvecTableau(const int t[], int n);
    int ajouterPatient(const Patient& p);
    void afficherStats() const;
};
*/

// Raffinement de Hopital
class Hopital
{
  public:
    Hopital(const string& nom, int nCh, int nlitsCh);
    void initialiserAvecTableau(const int t[], int n);
    int ajouterPatient(const Patient& p);
    void afficherStats() const;
```

```
    string getNom() const;
    int getNbChambres() const;
    int getNbChPleines() const;
    int getNbChVides() const;
    double tauxOccup() const;

    bool chambreDispo(int nCh) const;
    int getChambreDispo() const;

    void afficher() const;
    void afficherOccupation() const;
    void afficherPatientsParChambre() const;
    void afficherTout() const;
  private:
    bool ajouterPatient(const Patient& p, int nCh);

    string m_nom;
    vector<Chambre> m_t;
    int m_nLits;
    int m_nLitsOcc;
    int m_nChVides;
    int m_nChPleines;
};

#include "Hopital.cpp"
#endif
```

```
#include <iostream>
#include <string>
using namespace std;

/**
  Constructeur de Hopital
  @param[in] nom - Nom de l'hopital
  @param[in] nCh - Nombre de chambres
  @param[in] nlitsCh - Nombre de lits par chambre
*/
Hopital::Hopital(const string& nom, int nCh, int nlitsCh)
: m_nom(nom),
  m_t(),
  m_nLits(nCh*nlitsCh),
  m_nLitsOcc(0),
  m_nChVides(nCh),
  m_nChPleines(0)
{
  // Retaille le vecteur des chambres
  m_t.resize(nCh);
  // Initialise avec des chambres de capacité nlitsCh
  for (int k = 0; k < nCh; ++k)
  {
    m_t[k] = Chambre(k, nlitsCh);
  }
}

// Accesseur du nom
string Hopital::getNom() const
{
  return m_nom;
```

```
}

// Accesseur du nombre de chambres
int Hopital::getNbChambres() const
{
  return m_t.size();
}

// Accesseur du nombre de chambres pleines
int Hopital::getNbChPleines() const
{
  return m_nChPleines;
}

// Accesseur du nombre de chambres vides
int Hopital::getNbChVides() const
{
  return m_nChVides;
}

// Teste si le numero de chambre nCh dispose d'une place
bool Hopital::chambreDispo(int nCh) const
{
  return (0 < nCh and nCh < getNbChambres() and not m_t[nCh].estPleine());
}

// Trouve la chambre la plus remplie
// Même algorithme que pour la solution procédurale
// @return indice de la chambre, -1 si toutes sont pleines
int Hopital::getChambreDispo() const
{
  int maxNonPlein = -1;
  int rs = -1;
  for (int k = 0; k < m_t.size(); ++k)
  {
    if (not m_t[k].estPleine() and m_t[k].getOccupation() > maxNonPlein)
    {
      rs = k;
      maxNonPlein = m_t[k].getOccupation();
    }
  }
  return rs;
}

// Ajoute un patient dans une chambre donnee
// @return Vrai si l'operation a ete effectuee
bool Hopital::ajouterPatient(const Patient& p, int nCh)
{
  if (not chambreDispo(nCh))
  {
    return false;
  }
  else
  {
    m_t[nCh].ajouterPatient(p);
    if (m_t[nCh].getOccupation() == 1)
    {
      --m_nChVides;
```

```
    }
    if (m_t[nCh].estPleine())
    {
      ++m_nChPleines;
    }
    ++m_nLitsOcc;
    return true;
  }
}

// Ajoute un patient
int Hopital::ajouterPatient(const Patient& p)
{
  int num = getChambreDispo();
  // Redifere l'operation
  return (ajouterPatient(p, num) ? num : -1);
}

// Conversion int vers string
string int2str(int n)
{
  string s;
  while (n != 0)
  {
   s = string(1, '0'+n%10)+s;
   n = n / 10;
  }
  return s;
}

// Initialise avec un tableau
// Les patients sont crées avec des noms p1,p2... et des numéros 1,2...
void Hopital::initialiserAvecTableau(const int t[], int n)
{
  string nom = "p";
  int ns = 1;
  bool b;
  for (int k = 0; k < n; ++k)
  {
    for (int j = 0; j < t[k]; ++j, ++ns)
    {
      Patient p = Patient(nom+int2str(ns), ns);
      b = ajouterPatient(p, k);
      if (not b)
      {
        cout<<"*** Pas de place dans la chambre "<<int2str(k)
            <<" pour le patient "<<nom<<int2str(ns)<<endl;
      }
    }
  }
}

// Taux d'occupation
double Hopital::tauxOccup() const
{
  return 1.0 * m_nLitsOcc / m_nLits;
}
```

```
void Hopital::afficher() const
{
  cout<<"Hopital "<<m_nom<<endl;
  cout<<"  Nb de chambres = "<<m_t.size()<<endl;
  cout<<"  Capacite en lits = "<<m_nLits<<endl;
}

void Hopital::afficherStats() const
{
  cout<<"  Nb lits occupes     = "<<m_nLitsOcc<<endl;
  cout<<"  Nb chambres vides   = "<<m_nChVides<<endl;
  cout<<"  Nb chambres pleines = "<<m_nChPleines<<endl;
  cout<<"  Taux d'occupation   = "<<tauxOccup()<<endl;
}

void Hopital::afficherOccupation() const
{
  cout<<"Nombre patients par chambre"<<endl;
  for (int k = 0; k < m_t.size(); ++k)
  {
    cout<<" Chambre "<<k<<" : "<<m_t[k].getOccupation()<<endl;
  }
}

void Hopital::afficherPatientsParChambre() const
{
  cout<<"Patients par chambre"<<endl;
  for (int k = 0; k < m_t.size(); ++k)
  {
    cout<<" Chambre "<<k<<" : ";
    m_t[k].afficher();
    cout<<endl;
  }
}

void Hopital::afficherTout() const
{
  afficher();
  afficherOccupation();
  afficherStats();
  afficherPatientsParChambre();
}
```

3.11 Codage de la classe Chambre

Une chambre sera représentée par :

- Le numéro de la chambre `numCh` (entier).
- La capacité en lits `nbLits` (entier).
- Le tableau des patients `tab` (`Vecteur<Patient>`).
- Le nombre de lits occupés `litsOcc` (entier).

Incluez les attributs.

Écrivez le constructeur par défaut ainsi que le constructeur normal paramétré par le numéro de la chambre et sa capacité.

Écrivez les accesseurs `getNumeroCh` du numéro de chambre, `getCapacite` de la capacité en lits et `getOccupation` du nombre de lits occupés.

Écrivez la méthode `estVide` qui teste et renvoie `Vrai` si la chambre est vide, `Faux` sinon.

De même, écrivez la méthode `estPleine` qui teste et renvoie `Vrai` si la chambre est pleine, `Faux` sinon.

Écrivez la méthode `ajouterPatient(p)` qui ajoute un `Patient p`. Elle renvoie `Vrai` si l'opération a été effectuée, `Faux` sinon (chambre pleine).

Enfin écrivez la méthode `afficher` qui affiche l'état de la chambre.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Chambre.hpp] @[Chambre.cpp]

```cpp
#ifndef CHAMBRE_CLASS
#define CHAMBRE_CLASS

#include <vector>
using namespace std;
#include "Patient.hpp"

/*
// Premier prototype de Chambre
class Chambre
{
  public:
    Chambre();
    Chambre(int num, int nlits);
    int getOccupation() const;
    bool estVide() const;
    bool estPleine() const;
    bool ajouterPatient(const Patient& p);
    void afficher() const;
};
*/

// Raffinement de Chambre
class Chambre
{
  public:
    Chambre();
    Chambre(int num, int nlits);
    int getOccupation() const;
```

```
    bool estVide() const;
    bool estPleine() const;
    bool ajouterPatient(const Patient& p);
    void afficher() const;

    int getNumeroCh() const;
    int getCapacite() const;
  private:
    int m_numCh; // numero de la chambre
    int m_nbLits; // capacite en lits
    vector<Patient> m_tab; // tableau des patients
    int m_litsOcc; // nombre de lits occupes
};
#include "Chambre.cpp"
#endif
```

```
// Constructeur par defaut (pour les tableaux)
Chambre::Chambre()
: m_numCh(0), m_nbLits(0), m_tab(), m_litsOcc(0)
{}

/**
  Constructeur normal
  @param[in] num - numero de chambre
  @param[in] nlits - nombre de lits
*/
Chambre::Chambre(int num, int nlits)
: m_numCh(num), m_nbLits(nlits), m_tab(nlits), m_litsOcc(0)
{}

// Accesseur du numero de chambre
int Chambre::getNumeroCh() const
{
  return m_numCh;
}

// Accesseur de la capacite en lits
int Chambre::getCapacite() const
{
  return m_nbLits;
}

// Accesseur du nombre de lits occupes
int Chambre::getOccupation() const
{
  return m_litsOcc;
}

// Test de chambre vide
bool Chambre::estVide() const
{
  return (getOccupation() == 0);
}

// Test de chambre pleine
bool Chambre::estPleine() const
{
  return (getOccupation() == getCapacite());
```

```
}

// Ajoute un patient
bool Chambre::ajouterPatient(const Patient& p)
{
  if (estPleine())
  {
    return false;
  }
  else
  {
    m_tab[m_litsOcc] = p;
    ++m_litsOcc;
    return true;
  }
}

void Chambre::afficher() const
{
  if (estVide())
  {
    cout<<"Aucun patient"<<endl;
  }
  else
  {
    cout<<endl;
    for (int k = 0; k < m_litsOcc; ++k)
    {
      m_tab[k].afficher();
    }
  }
}
```

3.12 Programme principal

Testez.

Validez votre programme avec la solution.

Solution C++ @[pghopitalC1.cpp]

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

#include "Hopital.hpp"
// Premiere version du programme
int main()
{
  int t[]={1,3,4,0,2,3,4,4,0,1};
  int nchambres = 10;

  Hopital h = Hopital("Necker", nchambres, 4);
```

```
  h.initialiserAvecTableau(t,nchambres);
  h.afficherStats();

  Patient a1 = Patient("Marie",12345);
  int chAff = h.ajouterPatient(a1);
  if (chAff < 0)
  {
    cout<<"Hopital plein -- Ajout impossible"<<endl;
  }
  else
  {
    cout<<"Patient dans la chambre "<<chAff<<endl;
  }
}
```

4 Références générales

Comprend [Carrez-AL1] ■