Liste chaînée d'individus [pn02] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 20 mai 2018

Table des matières

C++ - Liste chaînée d'individus (Solution)

Mots-Clés Gestion dynamique de mémoire, Liste, Classe ■
Difficulté ●●○ (2 h) ■

Objectif
Cet exercice réalise un extrait de liste linéaire doublement chainée (type `Liste`) de chaînes de caractères (type `Element`). La classe ne définit que les méthodes minimales pour sa modification.

EXTENSION DU PROGRAMME juin 2017 – par des fonctions estvide, taille, etc. ■

1 Liste chaînée d'individus

1.1 Classe Element

La classe `Element` définit le type des éléments dans la liste chaînée.

Définissez une classe `Element` comprenant un champ `info` de type chaîne de caractères.

Écrivez un constructeur initialisant son attribut.

Écrivez un accesseur `toString` qui renvoie la valeur de son attribut.

Validez votre classe avec la solution.

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

```
#ifndef ELEMENT_CLASS
#define ELEMENT_CLASS
#include <string>
using namespace std;

/** Type des elements */
class Element
{
  public:
    Element(const string& s)
    : m_info(s)
    {
    }

    string toString() const
    {
      return m_info;
    }

  private:
    string m_info;
};
#endif
```

1.2 Classe Noeud

La classe `Noeud` définit un noeud de la liste.

Définissez une classe `Noeud` comprenant :

- Une valeur `donnee` de type `Element`.
- Un pointeur `suiv` vers le `Noeud` suivant.
- Un pointeur `prev` vers le `Noeud` précédent.

Écrivez les constructeurs :

- Le constructeur par défaut.
- Le constructeur normal (à trois paramètres) initialisant les attributs.

Écrivez les accesseurs :

- `getDonnee` de la donnée.
- `getSuiv` du pointeur vers l'élément suivant.
- `getPrev` du pointeur vers l'élément précédent.

Écrivez les mutateurs :

- `setDonnee(s)` qui fixe la valeur de la donnée.
- `setSuiv(suiv)` qui fixe le pointeur vers l'élément suivant.
- `setPrev(prev)` qui fixe le pointeur vers l'élément précédent.

Validez votre classe et vos méthodes avec la solution.

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

```cpp
#ifndef NOEUD_CLASS
#define NOEUD_CLASS
#include <string>
using namespace std;

/** Noeud de la liste */
#include "Element.hpp"
class Noeud
{
  public:
    Noeud(const Element& s);
    Noeud(const Element& s, Noeud* suiv, Noeud* prev);
    Element getDonnee() const;
    Noeud* getSuiv() const;
    Noeud* getPrev() const;
    void setDonnee(const Element& s);
    void setSuiv(Noeud* suiv);
    void setPrev(Noeud* prev);
  private:
    Element m_donnee;  // valeur dans le noeud
    Noeud *m_suiv; // pointeur vers element suivant
    Noeud *m_prev; // pointeur vers element precedent
};
#include "Noeud.cpp"
#endif
```

```
/**
  Constructeur par defaut
  @param[in] s - valeur d'initialisation
*/
Noeud::Noeud(const Element& s)
: m_donnee(s), m_suiv(NULL), m_prev(NULL)
{}

/**
  Constructeur normal
  @param[in] s - valeur d'initialisation
  @param[in] suiv - pointeur sur le noeud suivant
  @param[in] prev - pointeur sur le noeud precedent
*/
Noeud::Noeud(const Element& s, Noeud* suiv, Noeud* prev)
: m_donnee(s), m_suiv(suiv), m_prev(prev)
{}

/**
  Accesseur de la donnee
  @return Valeur de la donnee
*/
Element Noeud::getDonnee() const
{
  return m_donnee;
}

/**
  Accesseur du pointeur vers element suivant
  @return pointeur vers element suivant
*/
Noeud* Noeud::getSuiv() const
{
  return m_suiv;
}

/**
  Accesseur du pointeur vers element precedent
  @return pointeur vers element precedent
*/
Noeud* Noeud::getPrev() const
{
  return m_prev;
}

/**
  Fixe la donnee
  @param[in] s - valeur d'initialisation
*/
void Noeud::setDonnee(const Element& s)
{
  m_donnee = s;
}

/**
  Fixe le pointeur vers l'element suivant
  @param[in] suiv - valeur d'initialisation
*/
```

```
void Noeud::setSuiv(Noeud* suiv)
{
  m_suiv = suiv;
}

/**
  Fixe le pointeur vers l'element precedent
  @param[in] prev - valeur d'initialisation
*/
void Noeud::setPrev(Noeud* prev)
{
  m_prev = prev;
}
```

1.3 Classe Iterateur

La classe `Iterateur` indique une position dans la liste ou au delà de la fin de la liste.

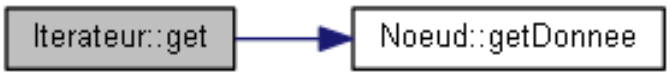
Définissez une classe `Iterateur` comprenant un pointeur `position` vers le `Noeud` de l'élément courant.

Écrivez les constructeurs :

- Le constructeur par défaut.
- Un constructeur normal (à un paramètre) qui initialise son attribut.

Écrivez les accesseurs :

- `base` de la position.
- `get` de la valeur du noeud pointé.



Écrivez un mutateur `set(s)` qui fixe la valeur du noeud pointé.

Écrivez les méthodes :

- `suivant` qui positionne l'`Iterateur` sur le noeud suivant.
- `precedent` qui positionne l'`Iterateur` sur le noeud précédent.

Écrivez une méthode `equals(b)` qui compare l'`Iterateur` courant avec un `Iterateur b` et qui renvoie `Vrai` s'ils définissent la même position, `Faux` sinon.

Validez votre classe et vos méthodes avec la solution.

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

```
#ifndef ITERATEUR_CLASS
#define ITERATEUR_CLASS

/**
  Indique une position dans la liste ou au dela de la fin de la liste
*/
#include "Noeud.hpp"
class Iterateur
{
  public:
    Iterateur();
    explicit Iterateur(Noeud* p);
    Noeud* base() const;
    Element get() const;
    void set(const Element& s);
    void suivant();
    void precedent();
    bool equals(Iterateur b) const;
  private:
   Noeud *m_position; // élément courant
};
#include "Iterateur.cpp"
#endif
```

```
/**
  Construit un Iterateur qui ne pointe a aucune liste
*/
Iterateur::Iterateur()
: m_position(NULL)
{}

/**
  Construit un Iterateur qui pointe sur un noeud
  @param[in] p - pointeur sur le noeud
*/
Iterateur::Iterateur(Noeud* p)
: m_position(p)
{}
```

```
/**
  Accesseur de la position
  @return la base de la position
*/
Noeud* Iterateur::base() const
{
  return m_position;
}

/**
  Valeur du noeud auquel l'Iterateur pointe
  @return Valeur du noeud auquel l'Iterateur pointe
*/
Element Iterateur::get() const
{
  assert(m_position != NULL);
  return m_position->getDonnee();
}

/**
  Remplace la valeur du noeud a la position donnée par l'iterateur
  @param[in] s - valeur d'initialisation
*/
void Iterateur::set(const Element& s)
{
  assert(m_position != NULL);
  m_position->setDonnee(s);
}

/**
  Avance l'Iterateur au noeud suivant
*/
void Iterateur::suivant()
{
  assert(m_position != NULL);
  m_position = m_position->getSuiv();
}

/**
  Recule l'Iterateur au noeud precedent
*/
void Iterateur::precedent()
{
  assert(m_position != NULL);
  m_position = m_position->getPrev();
}

/**
  Compare deux iterateurs
  @param[in] b - un Iterateur
  @return Vrai si même position
*/
bool Iterateur::equals(Iterateur b) const
{
  return m_position == b.m_position;
}
```

1.4 Classe Liste

La classe `Liste` définit la liste linéaire doublement chaînée d'éléments de type `Element`.

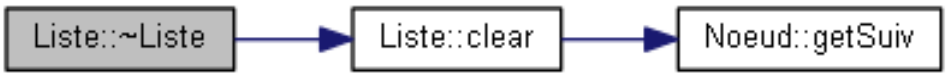
Définissez une classe `Liste` comprenant :

- Un pointeur `premier` sur le `Noeud` du premier élément.
- Un pointeur `dernier` sur le `Noeud` du dernier élément.

Écrivez le constructeur par défaut qui construit une `Liste` vide.

Écrivez une méthode `clear()` qui détruit chacun des éléments de la liste.

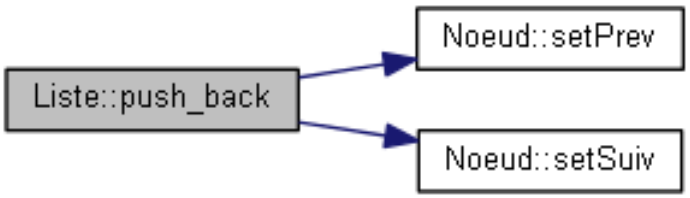
Déduisez le destructeur qui détruit la `Liste`.



Écrivez une méthode `begin()` qui renvoie un `Iterateur` pointant au début de la `Liste`.

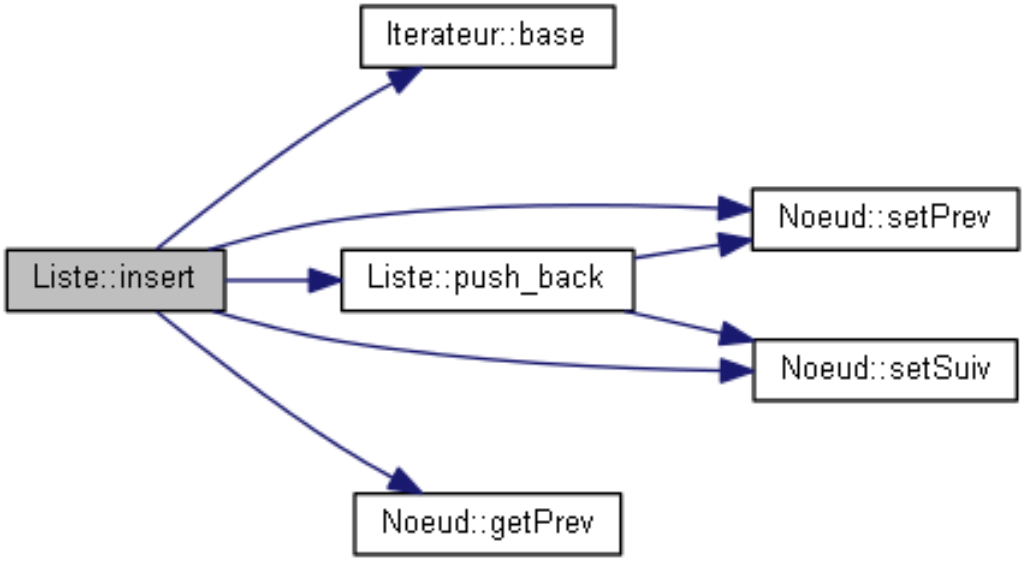
De même, écrivez une méthode `end()` qui renvoie un `Iterateur` pointant **après** la fin de la `Liste`.

Écrivez une méthode `push_back(e)` qui ajoute un `Element e` en fin de liste.



Aide simple
Elle crée un `Noeud` puis met à jour les liens selon que la liste est vide ou non.

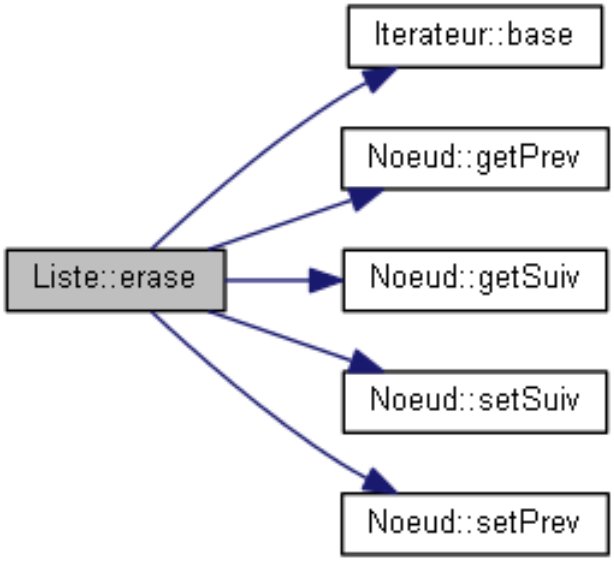
Écrivez une méthode `insert(it,e)` qui insère un `Element e` dans la liste **avant** la position de l'`Iterateur it` (c.-à-d. le nouvel élément prend la position de l'élément en position `it`).



Aide simple

La méthode teste le cas d'une insertion en fin de liste ; sinon elle insère un élément dans la liste et met à jour les chaînages.

Écrivez une méthode `erase(it)` qui supprime l'élément de la liste repéré par l'`Iterateur it`. Elle renvoie l'`Iterateur` pointant sur l'élément suivant de l'élément supprimé.



Validez votre classe et vos méthodes avec la solution.

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

```
#ifndef LISTE_CLASS
#define LISTE_CLASS
#include <iostream>
#include <string>
#include <cassert>
using namespace std;
#include "Noeud.hpp"
#include "Iterateur.hpp"
/**
  (Extrait) Liste lineaire doublement chainee
*/
class Liste
{
  public:
    Liste();
    ~Liste();
    void clear();
    Iterateur begin();
    Iterateur end();
    void push_back(const Element& elem);
    void insert(Iterateur& it, const Element& elem);
    Iterateur erase(Iterateur it);
  private:
    Noeud *m_premier; // pointeur vers premier element
    Noeud *m_dernier; // pointeur vers dernier element
};
#include "Liste.cpp"
#endif
```

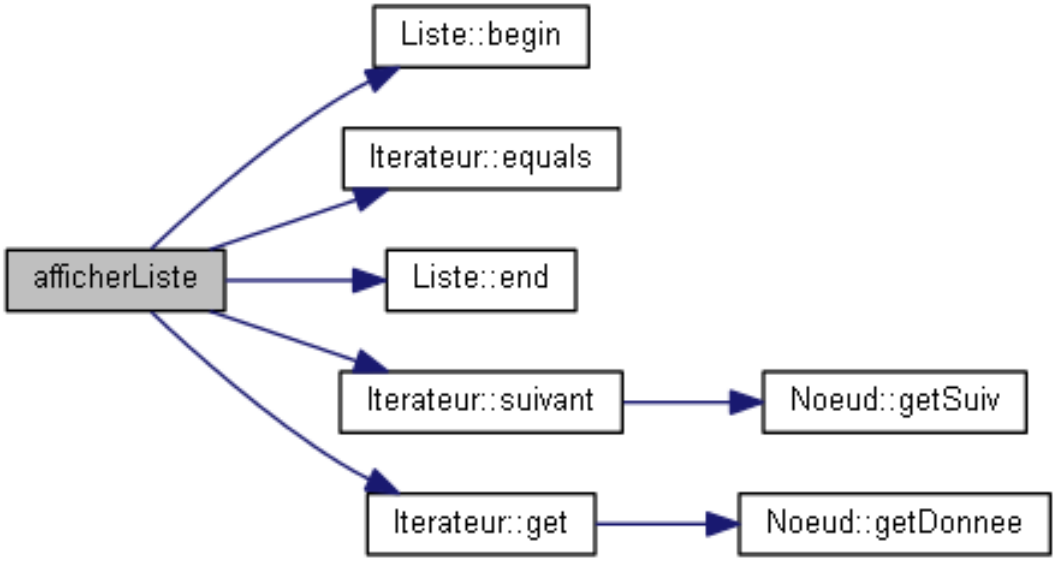
```
/**
  Construit une liste vide
*/
Liste::Liste()
: m_premier(NULL), m_dernier(NULL)
{}

/**
  Détruit la liste
*/
Liste::~Liste()
{
  clear();
}

/**
  Purge la liste
*/
void Liste::clear()
{
  for (Noeud *a_enlever = m_premier; a_enlever != NULL; a_enlever = a_enlever->getSuiv())
  {
    delete a_enlever;
  }
  m_premier = NULL;
  m_dernier = NULL;
}
```

```
/**
  Iterateur de debut de liste
  @return Iterateur pointant au début de la liste
*/
Iterateur Liste::begin()
{
  return Iterateur(m_premier);
}

/**
  Iterateur de fin de liste
  @return Iterateur pointant apres la fin de la liste
*/
Iterateur Liste::end()
{
  return Iterateur();
}

/**
  Ajoute un element en fin de liste
  @param[in] elem - un Element
*/
void Liste::push_back(const Element& elem)
{
  Noeud* nouveau = new Noeud(elem);
  // cas liste vide ou non
  if (m_dernier == NULL)
  {
    m_premier = nouveau;
    m_dernier = nouveau;
  }
  else
  {
    nouveau->setPrev(m_dernier);
    m_dernier->setSuiv(nouveau);
    m_dernier = nouveau;
  }
}

/**
  Insere un element dans la liste avant position it
  @param[in,out] it - un Iterateur
  @param[in] elem - un Element
*/
void Liste::insert(Iterateur& it, const Element& elem)
{
  if (it.base() == NULL)
  {
    push_back(elem);
    it = Iterateur(m_dernier);
  }
  else
  {
    Noeud *apres = it.base();
    Noeud *avant = apres->getPrev();
    Noeud *nouveau = new Noeud(elem, apres, avant);
    apres->setPrev(nouveau);
    // Insertion au debut ou ailleurs
```

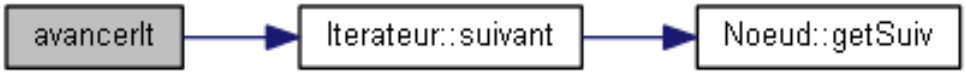
```
    if (avant == NULL)
    {
      m_premier = nouveau;
    }
    else
    {
      avant->setSuiv(nouveau);
    }
  }
}

/**
  Supprime l'element de la liste repere par it
  @param[in,out] it1 - un Iterateur
  @return Iterateur pointant à l'element suivant l'element efface
*/
Iterateur Liste::erase(Iterateur it1)
{
  Iterateur it = it1;
  assert(it.base() != NULL);
  Noeud *a_enlever = it.base();
  Noeud *avant = a_enlever->getPrev();
  Noeud *apres = a_enlever->getSuiv();
  if (a_enlever == m_premier)
  {
    m_premier = apres;
  }
  else
  {
    avant->setSuiv(apres);
  }
  if (a_enlever == m_dernier)
  {
    m_dernier = avant;
  }
  else
  {
    apres->setPrev(avant);
  }
  it = Iterateur(apres);
  delete a_enlever;
  return it;
}
```

1.5 Programme de test

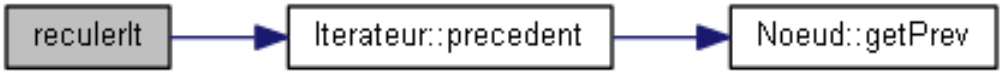
Écrivez une procédure `afficherListe(txt,t)` qui affiche un texte `txt| (chaine de caractères) puis les valeurs d'une \lstinline`Liste t@.



Écrivez une procédure `avancerIt(it,k)` qui avance un `Iterateur it` de `k` (entier) positions.



De même, écrivez une procédure `reculerIt(it,k)` qui recule un `Iterateur it` de `k` (entier) positions.



Écrivez un programme qui instancie une `Liste etudiants`, ajoute les six éléments suivants puis l'affiche.

```
Bon, Jean
Fine, Louis
Gat, Rene
Stant, Alain
Tard, Guy
Tron, Paul
```

Déclarez un `Iterateur it`, ajoutez l'élément suivant en position 3 puis affichez la nouvelle `Liste`.

```
Fere, Lucie
```

Supprimez l'élément en position 4 puis affichez la `Liste`.

Ajoutez l'élément suivant en fin de liste puis affichez la nouvelle `Liste`.

```
Kaderate, Yamamoto
```

Modifiez l'élément en position -3 par la valeur suivante puis affichez la `Liste`.

```
Goland, Henri
```

Testez. Résultat d'exécution :

```
==> Apres creation 6 elements
Bon, Jean
Fine, Louis
Gat, Rene
Stant, Alain
Tard, Guy
Tron, Paul

==> Apres insertion d'un element en position 3
Bon, Jean
Fine, Louis
Fere, Lucie
Gat, Rene
Stant, Alain
Tard, Guy
Tron, Paul

==> Apres suppression de l'element en position 4
Bon, Jean
Fine, Louis
Fere, Lucie
Stant, Alain
Tard, Guy
Tron, Paul

==> Apres ajout en fin de liste via insert
Bon, Jean
Fine, Louis
Fere, Lucie
Stant, Alain
Tard, Guy
Tron, Paul
Kaderate, Yamamoto

==> Apres modification de l'element en position -3
Bon, Jean
Fine, Louis
Fere, Lucie
```

```
Stant, Alain
Goland, Henri
Tron, Paul
Kaderate, Yamamoto
```

Validez vos procédures et votre programme avec la solution.

Solution C++ @[pgliste.cpp]

```cpp
#include <iostream>
#include <string>
#include <cassert>
using namespace std;
#include "Liste.hpp"
/**
  Affiche un texte puis les valeurs d'une liste
  @param[in] txt - un texte
  @param[in] t - une Liste
*/
void afficherListe(const string& txt, /*const*/ Liste& t)
{
  cout << "==> " << txt << endl;
  for (Iterateur it = t.begin(); !it.equals(t.end()); it.suivant())
  {
    cout<<it.get().toString()<<endl;
  }
  cout<<endl;
}

/**
  Avance un itérateur de k positions
  @param[in,out] it - un Iterateur
  @param[in] k - nombre de positions
*/
void avancerIt(Iterateur& it, int k)
{
  for (int j = 1; j <= k; ++j)
  {
    it.suivant();
  }
}

/**
  Recule un iterateur de k positions
  @param[in,out] it - un Iterateur
  @param[in] k - nombre de positions
*/
void reculerIt(Iterateur& it, int k)
{
  for (int j = 1; j <= k; ++j)
  {
    it.precedent();
  }
}
```

```
/**
  Procédure de test
*/
int main()
{
  Liste etudiants;
  Iterateur it;
  etudiants.push_back(Element("Bon, Jean"));
  etudiants.push_back(Element("Fine, Louis"));
  etudiants.push_back(Element("Gat, Rene"));
  etudiants.push_back(Element("Stant, Alain"));
  etudiants.push_back(Element("Tard, Guy"));
  etudiants.push_back(Element("Tron, Paul"));
  afficherListe("Apres creation 6 elements", etudiants);
  it = etudiants.begin();
  avancerIt(it, 2);
  etudiants.insert(it, Element("Fere, Lucie"));
  afficherListe("Apres insertion d'un element en position 3", etudiants);
  it = etudiants.begin();
  avancerIt(it, 3);
  it = etudiants.erase(it);
  afficherListe("Apres suppression de l'element en position 4", etudiants);
  it = etudiants.end();
  etudiants.insert(it, Element("Kaderate, Yamamoto"));
  afficherListe("Apres ajout en fin de liste via insert", etudiants);
  reculerIt(it, 2);
  it.set(Element("Goland, Henri"));
  afficherListe("Apres modification de l'element en position -3", etudiants);
}
```

1.6 Fonctions

Écrivez une fonction `estvide(t)` qui teste et renvoie `Vrai` si une `Liste t` est vide, `Faux` sinon.

Écrivez une fonction `taille(t)` qui calcule et renvoie la taille d'une `Liste t`.

2 Références générales

Comprend [Chappelier-CPP1 :c4 :et] ■