Programmation Orientée Objet C++ — Episode #6

TP4 — Correction partielle

Les noeuds

Représentation ?

- Structure envisageable :

```c
struct noeud {
  int val;
  struct noeud* fils_gauche;
  struct noeud* fils_droit;
};
```

- … mais moins pratique pour la suite.

La classe Noeud

Les attributs, constructeurs, destructeur

```
1  class Noeud {
2    private:
3      int val;
4      Noeud* fg;
5      Noeud* fd;
6      ...
7    public:
8      Noeud();
9      Noeud(int _val);
10     Noeud(const Noeud &n);
11     ~Noeud(); // Suffisant ?
12 };
```

La classe `Noeud`

Les accesseurs

```
1  Noeud* get_fg() const { return fg; }
2  Noeud* get_fd() const { return fd; }
3  int get_val() const { return val; }
```

La classe `Noeud`

Les mutateurs

```
1  void set_val(int _val) { val = _val; }
2  void set_fg(Noeud* n) { fg = n; }
3  void set_fd(Noeud* n) { fd = n; }
4  void set_pere(Noeud* n) { pere = n; }
```

Les arbres binaires

Un arbre binaire ?

Structure clairement définie. MAIS :

- Pour l'exemple : classe abstraite.
- La suppression sera redéfinie dans les classes Filles.

La classe `ArbreBin`

Les attributs, constructeurs, destructeur :

```
1  class ArbreBin {
2    // Pour accès depuis classe Fille
3    protected:
4      Noeud* racine;
5    public:
6      ArbreBin();
7      ArbreBin(int _val);
8      // Héritage : destructeur virtuel
9      virtual ~ArbreBin();
```

Les méthodes communes

Plusieurs méthodes peuvent être définies dans cette classe (sous certaines hypothèses) :

- Affichage
- Recherche
- Suppression d'une feuille
- Suppression d'un noeud avec un seul fils
- Insertion

Les méthodes communes

```
1  Noeud* get_racine() const { return racine; }
2
3  // Méthodes propres à Arbre binaire
4  void afficher(Noeud* n);
5  void supprimer_feuille(Noeud* n);
6  void supprimer_un_fils(Noeud* n);
7
8  // Méthodes virtuelles --- redéfinition possible
9  virtual Noeud* rechercher(Noeud *n, int _val);
10 virtual bool inserer(int _val);
11
12 // Méthodes virtuelles pures --- redéfinition obligatoire
13 virtual bool inserer(Noeud* &ins, Noeud* pere, int _val) = 0;
14 virtual bool supprimer(int _val) = 0;
```

afficher

```
1  void ArbreBin::afficher(Noeud* n) {
2    if(n == NULL) { return;}
3    cout << n->get_val() << " ";
4    if(!n->get_fg() && !n->get_fd()) { return; }
5    cout << "( ";
6    afficher(n->get_fg());
7    cout << " )";
8    if(n->get_fd()) {
9      cout << "( ";
10     afficher(n->get_fd());
11     cout << " )";
12   }
13 }
```

inserer

```
1  bool ArbreBin::inserer(int _val) {
2    Noeud* tmp = racine;
3    Noeud* pere;
4    while(tmp != NULL) {
5      pere = tmp;
6      tmp = tmp->get_fg();
7    }
8    tmp = new Noeud(_val);
9    tmp->set_pere(pere);
10   pere->set_fg(tmp);
11 }
```

rechercher — ?

```
1  static Noeud* rechercher_rec(Noeud *n, int _val) {
2    // EXERCICE
3  }
4
5  Noeud* ArbreBin::rechercher(int _val) {
6    return rechercher_rec(racine, _val);
7  }
```

Autre solution

Passer le noeud en paramètre permet de rechercher dans un sous-arbre :

```
1  Noeud* ArbreBin::rechercher(const Noeud* n, int _val) {
2    // EXERCICE
3  }
```

supprimer_feuille

```
1  void ArbreBin::supprimer_feuille(Noeud* n) {
2    assert(n != NULL);
3    Noeud* pere = n->get_pere();
4    if(n == pere->get_fg())
5      pere->set_fg(NULL);
6    else
7      pere->set_fd(NULL);
8      delete n;
9      n = NULL; // Correct ?
10 }
```

supprimer_feuille

```
1  void ArbreBin::supprimer_feuille(Noeud* &n) {
2    assert(n != NULL);
3    Noeud* pere = n->get_pere();
4    if(n == pere->get_fg())
5      pere->set_fg(NULL);
6    else
7      pere->set_fd(NULL);
8      delete n;
9      n = NULL;
10 }
```

supprimer_un_fils

```
1  void ArbreBin::supprimer_un_fils(Noeud* n) {
2    assert(n != NULL);
3    Noeud *pere = n->get_pere();
4    if(n->get_fg())
5      n = n->get_fg();
6    else
7      n = n->get_fd();
8    n->set_pere(pere);
9    if(tmp == pere->get_fg())
10     pere->set_fg(n);
11   else
12     pere->set_fd(n);
13   delete n;
14 }
```

`supprimer_un_fils`

Complet ?

supprimer_un_fils

```
1  void ArbreBin::supprimer_un_fils(Noeud* &n) {
2    assert(n != NULL);
3    Noeud* fg = n->get_fg();
4    Noeud *fd = n->get_fd();
5    Noeud *pere = n->get_pere();
6    if(fg) {
7      *n = *fg;
8      delete fg;
9    }
10   else {
11     *n = *fd;
12     delete fd;
13   }
14   n->set_pere(pere);
15 }
```

Les arbres binaires de recherche

Définition

- Arbres binaires avec un **ordre** sur les clés
- équilibrés ?

La classe ArbreBinRec

Attributs ? Constructeurs ? Destructeur ?

```
1  class ArbreBinRec : public ArbreBin {
2    public:
3      ArbreBinRec();
4      ArbreBinRec(int _val);
5      ~ArbreBinRec();
6
7      // Méthodes virtuelles (pures ou non) redéfinies
8      Noeud* rechercher(const Noeud* n, int _val);
9      bool inserer(int _val);
10     bool inserer(Noeud* &ins, Noeud *n, int _val);
11     bool supprimer(int _val);
12 };
```

L'insertion — #1

```
1  bool ArbreBin::inserer(int _val) {
2    if(rechercher(racine, _val)) return false;
3    Noeud *tmp = racine;
4    Noeud *pere_tmp;
5
6    while(tmp != NULL) {
7      pere_tmp = tmp;
8      if(tmp->get_val() == _val)
9        return false;
10     else if (tmp->get_val() > _val)
11       tmp = tmp->get_fg();
12     else
13       tmp = tmp->get_fd();
14   }
15   tmp = new Noeud(_val);
16   tmp->set_pere(pere_tmp);
17   if(pere_tmp->get_val() > _val)
18     pere_tmp->set_fg(tmp);
19   else
20     pere_tmp->set_fd(tmp);
21   return true;
```

L'insertion — #2

Et si on modifiait directement les noeuds ?

```
1  bool ArbreBinRec::inserer(Noeud* &ins, Noeud* pere, int _val) {
2    if(rechercher(racine, _val)) return false;
3    if(ins == NULL) {
4      ins = new Noeud(_val);
5      ins->set_pere(pere);
6      return true;
7    }
8    else if(ins->get_val() > _val)
9      inserer(ins->get_fg(), ins, _val);
10   else
11     inserer(ins->get_fd(), ins, _val);
12   return false;
13 }
```

Fonctionne ?

Ajout d'accesseurs !

```
1  Noeud* &get_fg() { return fg; }
2  Noeud* &get_fd() { return fd; }
3  Noeud* &get_pere() { return pere; }
```

Une référence sur un pointeur ?

Similaire à l'opérateur `[]` (autre solution)

Référence de pointeur ?

Voir tableau.

La suppression

Tout refaire ?

```
1  bool ArbreBinRec::supprimer(int _val) {
2    Noeud *n = rechercher(racine, _val);
3    // Différentes actions selon le(s) cas
4    if(!n) { return false; }
5    else if(!n->get_fg() && !n->get_fd())
6      supprimer_feuille(n);
7    else if(!n->get_fg() || !n->get_fd())
8      supprimer_un_fils(n);
9    else
10     supprimer_noeud(*this, n);
11   return true;
12 }
```

La suppression

Tout refaire ?

`supprimer_noeud ?`

La suppression

Tout refaire ?

```
1  static void supprimer_noeud(ArbreBinRec &a, Noeud *n) {
2    if(n == NULL) { return; }
3    // Fils droit le plus à gauche
4    Noeud *tmp = n->get_fd();
5    while(tmp->get_fg() != NULL)
6      tmp = tmp->get_fg();
7    // Permutation
8    int c = n->get_val();
9    n->set_val(tmp->get_val());
10   tmp->set_val(c);
11   // Suppression
12   if(tmp->get_fd())
13     a.supprimer_un_fils(tmp);
14   else
15   a.supprimer_feuille(tmp);
16 }
```

Autres possibilités

- Passer un noeud en paramètre -> `static` devient méthode de classe
- S'affranchir des mutateurs/accesseurs ?
- **Classe amie !**

ArbreBinRec friend de Noeud

```
1  bool ArbreBinRec::inserer(Noeud* &ins, Noeud* pere, int _val) {
2    if(rechercher(_val)) return false;
3    if(ins == NULL) {
4      ins = new Noeud(_val);
5      ins->get_pere() = pere;
6      return true;
7    }
8    else if(ins->val > _val) {
9      inserer(ins->fg, ins, _val);
10   }
11   // suppose qu'on n'insere pas une valeur deja presente
12   else {
13     inserer(ins->fd, ins, _val);
14   }
15   return false;
16 }
```

ArbreBinRec friend de Noeud

```
1  bool ArbreBinRec::inserer(Noeud* &ins, Noeud* pere, int _val) {
2    if(rechercher(_val)) return false;
3    if(ins == NULL) {
4      ins = new Noeud(_val);
5      ins->get_pere() = pere;
6      return true;
7    }
8    else if(ins->val > _val) {
9      inserer(ins->fg, ins, _val);
10   }
11   // suppose qu'on n'insere pas une valeur deja presente
12   else {
13     inserer(ins->fd, ins, _val);
14   }
15   return false;
16 }
```

Attention : les amis ne sont pas hérités !

ArbreBinRec friend de Noeud

```
1  bool ArbreBinRec::inserer(Noeud* &ins, Noeud* pere, int _val) {
2    if(rechercher(_val)) return false;
3    if(ins == NULL) {
4      ins = new Noeud(_val);
5      ins->get_pere() = pere;
6      return true;
7    }
8    else if(ins->val > _val) {
9      inserer(ins->fg, ins, _val);
10   }
11   // suppose qu'on n'insere pas une valeur deja presente
12   else {
13     inserer(ins->fd, ins, _val);
14   }
15   return false;
16 }
```

Seules les méthodes de **classe** sont concernées

ArbreBinRec friend de Noeud

```
1  bool ArbreBinRec::inserer(Noeud* &ins, Noeud* pere, int _val) {
2    if(rechercher(_val)) return false;
3    if(ins == NULL) {
4      ins = new Noeud(_val);
5      ins->get_pere() = pere;
6      return true;
7    }
8    else if(ins->val > _val) {
9      inserer(ins->fg, ins, _val);
10   }
11   // suppose qu'on n'insere pas une valeur deja presente
12   else {
13     inserer(ins->fd, ins, _val);
14   }
15   return false;
16 }
```

Rajouter dans Noeud :

```
1  friend class ArbreBin;
2  friend class ArbreBinRec;
```

La destruction

Où, quand et comment détruire ?

- Destruction d'un `Noeud` ?
- Destruction d'un `ArbreBin` ?
- Destruction d'un `ArbreBinRec` ?

Comment détruire ?

Exercice

Les arbres rouge et noir ?