Algorithmique et Programmation C++

TD 5

```
/* Classe de Liste chainee         *\
\* Par Gael Even et Jeremie Osmont */

#include <iostream.h>
```

Maillon de base

```
struct Maillon
{
        int info;
        Maillon * suiv;
};
```

Définition de la classe Liste

```
class Liste
{
        // Pointeur vers le premier maillon :
        Maillon * debut;
public:
        // Definition du constructeur :
        Liste() {debut=NULL;}
        // Declaration des fonctions membres :
        ~Liste();
        bool chercher(int);
        Maillon * chercher_bis(int);
        void inserer(int);
        void inserer_bis(int);
        bool supprimer(int);
        int cardinal();
        void affiche();
        void concatener(Liste &);
        Liste recopier();
        Liste recopier_bis();
};
```

Code du destructeur

```
Liste::~Liste()
{
        Maillon * p;
        Maillon * courant=debut;
        while(courant!=NULL)
        {
                p=courant;
                courant=courant->suiv;
                delete p;
        }
}
```

Recherche d'un maillon

```
bool Liste::chercher(int x)
{
        Maillon * m=debut;
        while(m!=NULL)
        {
                if(m->info==x) return true;
                m=m->suiv;
        }
        return false;
}
```

Recherche élaborée d'un maillon

```
Maillon * Liste::chercher_bis(int x)
{
        Maillon * m=debut;
        while (m!=NULL)
        {
                if (m->info==x) return m;
                m=m->suiv;
        }
        return NULL;
}
```

Insertion en tête d'un nouveau maillon

```
void Liste::inserer(int x)
{
        // Puis le cree :
        Maillon * m=new Maillon;
        // Le configure :
        m->info=x;
        // L'insere dans la chaine :
        m->suiv=debut;
        debut=m;
}
```

Suppression d'un maillon donné

```
bool Liste::supprimer(int x)
{
        Maillon * m=debut;
        Maillon * tmp;
        // Suppression de l'element de tete :
        if (debut->info==x)
        {
                debut=m->suiv;
                delete m;
                return true;
        }
        // Cas general - recherche du maillon precedent
        // celui a effacer :
        while ((m->suiv)&&(m->suiv->info!=x)) m=m->suiv;
        // Si l'element est present :
        if (m->suiv->info==x)
        {
                // Modifie les liens :
                tmp=m->suiv;
                // Ceci marche meme si c'est le dernier maillon
                m->suiv=m->suiv->suiv;
                // Puis l'efface :
                delete tmp;
                return true;
        }
        // Si on arrive ici, alors
        // l'element n'est pas present :
        return false;
}
```

Renvoi du nombre d'éléments de la liste

```
int Liste::cardinal()
{
        Maillon *m=debut;
        // Initialise le compteur :
        int inc=0;
        // Parcours la liste jusqu'a la fin :
        while(m!=NULL)
        {
                inc++;
                m=m->suiv;
        }
        // Puis renvoie le resultat :
        return inc;
}
```

Affichage de la totalité de la liste chaînée

```
void Liste::affiche()
{
        Maillon *m=debut;
        // Meme structure que pour cardinal() :
        while(m!=NULL)
        {
                cout << m->info << ";";
                m=m->suiv;
        }
        cout << endl;
        // Ceci n'est pas demande, mais tres pratique...
}
```

Concaténation de deux listes chaînées

```
void Liste::concatener(Liste & l2)
{
        Maillon * m=debut;
        // Verifie que la liste 1 n'est pas vide :
        if(debut==NULL)
                debut=l2.debut;
        else
        {
                // Cherche la fin de liste :
                while(m->suiv)  m=m->suiv;
                // 'Connecte' la liste 2 :
                m->suiv=l2.debut;
        }
        // Vide la liste 2 pour eviter les erreurs !
        l2.debut=NULL;
}
```

Insertion d'un maillon a sa place

```
void Liste::inserer_bis(int x)
{
        // Cree le nouveau maillon et le configure :
        Maillon * nm = new Maillon;
        nm->info=x;
        Maillon * m = debut;
        // Cas particulier - La liste est vide ou bien
        // l'element doit etre place en premier :
        if(!debut||debut->info>x)
        {
                nm->suiv=debut;
                debut=nm;
                return;
        }
        // Sinon cas general : recherche le maillon precedent :
        while ((m->suiv)&&(m->suiv->info <x)) m=m->suiv;
        // Puis modifie les pointeurs pour inserer le nouveau
        // maillon proprement dit :
        nm->suiv=m->suiv;
        m->suiv=nm;
}
```

Recopie de la liste

```
Liste Liste::recopier()
{
        // Cree une nouvelle liste :
        Liste l;
        // Cas 'piege' ou on veut copier une liste vide :
        if(debut==NULL) return l;
        // Cree quelques pointeurs de maillon :
        Maillon * m=debut;
        Maillon * tmp;
        Maillon * tmpprec;
        // Parcours la liste depuis le debut jusqu'a la fin :
        while(m!=NULL)
        {
                // Recopie 'a la main' :
                tmp = new Maillon;
                // Gere le cas du premier maillon :
                if (!l.debut) l.debut=tmp;
                tmp->info=m->info;
                // Ceci pour connecter les maillons entre eux :
                tmpprec->suiv=tmp;
                tmpprec=tmp;
                // Passe au suivant :
                m=m->suiv;
        }
        // N'oublie pas d'indiquer la fin de la liste :
        tmp->suiv=NULL;
        // Dans ce cas, le constructeur par recopie
        // par defaut fonctionne bien :
        return l;
}
```

Recopie inverse

```
Liste Liste::recopier_bis()
{
        // Cree la nouvelle liste :
        Liste l;
        Maillon * m=debut;
        while(m)
        {
                // Plus simple : on utilise l'insertion de tete :
                l.inserer(m->info);
                m=m->suiv;
        }
        // La aussi, le constructeur par recopie fait l'affaire :
        return l;
}
```

Programme de test

```
int main()
{
        // Cree un Liste (test du constructeur) :
        Liste l;

        // Test de l'insertion de base et de la recherche :
        cout << "Entrez une serie d'entiers separes par ENTREE :"
             << "\n0 pour arreter !\n";
        int ent=1;
        while(ent)
        {
                cout << ">"; cin >> ent;
                if (ent) if (l.chercher(ent))
                        cout << "Attention : " << ent
                             << " est deja present !";
                else
                {
                        l.inserer_bis(ent);
                        cout << "Valeur " << ent
                             << " enregistree.";
                }
                cout << endl;
        }
        cout << "\nNombre de valeurs entrees : "
             << l.cardinal() << " : ";
        l.affiche();

        // Test de la suppression :
        cout << "\nEntrez les nombres a supprimer :"
             << "\n0 pour arreter !\n";
        ent=1;
        while(ent)
        {
                cout << ">"; cin >> ent;
                if (ent) if (l.supprimer(ent))
                     cout << "Suppression OK";
                else cout << "Non present !";
                cout << "\nValeurs restantes : "
                     << l.cardinal() << " : ";
                l.affiche();
        }

        // Test de l'insertion dans une liste triee :
        Liste lb;
        cout << "\nEntrez une autre serie d'entiers :"
             << "\n0 pour arreter !\n";
        ent=1;
        while(ent)
        {
                cout << ">"; cin >> ent;
                if (ent) if (lb.chercher(ent))
                        cout << "Attention : " << ent
                             << " est deja present !";
                else
                {
                        lb.inserer_bis(ent);
                        cout << "Valeur " << ent
                             << " enregistree.";
                }
                cout << endl;
        }
        cout << "Nombre de valeurs entrees : "
             << lb.cardinal() << " : ";
        lb.affiche();
        cout << "MIRACLE : elles sont triees !\n";

        // Test de la concatenation :
        cout << "\nConcatenation des deux listes : ";
        l.concatener(lb);
        cout << "\nNombre total de valeurs : "
             << l.cardinal() << " : ";
        l.affiche();

        // Test de la recopie inversee :
        cout << "\nRecopie inverse :";
        Liste ld=l.recopier_bis();
        cout << "\nNombre de valeurs copiees : "
             << ld.cardinal() << " : ";
        ld.affiche();

        // Test de la recopie de base :
        cout << "\nCopie conforme :";
        Liste lc=l.recopier();
        cout << "\nNombre de valeurs copiees : "
             << lc.cardinal() << " : ";
        lc.affiche();

        cout << endl;

        // Fin du programme (test du destructeur !)
}
```