

# Implantation des TAD en C

## Exemple utilisant la SDD liste chaînée

Nicolas Delestre

# Plan

---

- 1 TAD polynôme
  - Analyse
  - Conception
- 2 Développement : première proposition
- 3 Développement : deuxième proposition
- 4 Conclusion

# Contexte

## Cahier des charges

- Développez une bibliothèque proposant le type Polynome

# Analyse

## Le TAD

<b>Nom:</b>	Polynome
<b>Utilise:</b>	<b>Naturel, Reel</b>
<b>Opérations:</b>	<p>polynome: <math>\rightarrow</math> Polynome</p> <p>obtenirDegre: Polynome <math>\rightarrow</math> <b>Naturel</b></p> <p>obtenirCoefficient: Polynome <math>\times</math> Naturel <math>\rightarrow</math> <b>Reel</b></p> <p>modifierCoefficient: Polynome <math>\times</math> <b>Reel</b> <math>\times</math> <b>Naturel</b> <math>\rightarrow</math> Polynome</p>
<b>Axiomes:</b>	<ul style="list-style-type: none"> <li>- <math>obtenirCoefficient(polynome(), i) = 0</math></li> <li>- <math>obtenirDegre(polynome()) = 0</math></li> <li>- <math>\forall i &gt; obtenirDegre(p), obtenirCoefficient(p, i) = 0</math></li> <li>- <math>obtenirCoefficient(p, obtenirDegre(p)) \neq 0</math></li> <li>- <math>obtenirCoefficient(modifierCoefficient(p, v, i), i) = v</math></li> </ul>

# Conception préliminaire

## Les signatures de fonctions procédures

- **fonction** polynome () : Polynome
- **fonction** obtenirDegre (p : Polynome) : **Naturel**
- **fonction** obtenirCoefficient (p : Polynome, degre : **Naturel**) : **Reel**
- **procédure** modifierCoefficient (**E/S** p : Polynome, **E** coef : **Reel**, degre : **Naturel**)

Nous pouvons aussi ajouter les fonctions (liées au choix du paradigme) suivantes :

- **fonction** sontEgaux (p1,p2 : Polynome) : **Booleen**
- **fonction** copier (p : Polynome) : Polynome

# Conception détaillée

## Représentation

**Type** Polynome = **Structure**

monomes : ListeChaine<Monome>

degres : **Naturel**

**finstructure**

## Attention

Il va donc falloir ajouter une opération de suppression

## Exercice

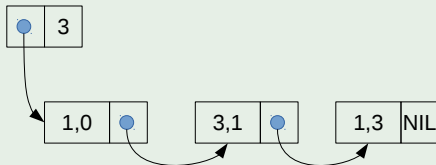
Spécifier et concevoir le TAD Monome

Il faut donc posséder la SDD ListeChaine, or par défaut le C ne la propose pas

# Première proposition : Développer une liste chaînée de monomes 1 / 8

- Cela implique :
  - Deux fichiers : `ListeChaineDeMonomes.c`, `ListeChaineDeMonomes.h`
  - Le choix du préfixe : `LCDM`
  - Le type `LCDM_ListeChaine`
  - L'ajout trois fonctions :
    - de copie d'une liste
    - de comparaison de deux listes
    - de suppression d'une liste

Exemple :  $x^3 + 3x + 1$



# Première proposition : Développer une liste chaînée de monomes 2 / 8

## ListeChaineDeMonomes.h

```

1 #ifndef __LISTE_CHAINEE_DE_MONOMES__
2 #define __LISTE_CHAINEE_DE_MONOMES__
3 #include <errno.h>
4 #include "Monome.h"
5
6 typedef struct LCDM_Noed* LCDM_ListeChaine;
7 typedef struct LCDM_Noed {
8     M_Monome IElement;
9     LCDM_ListeChaine listeSuiivante;
10 } LCDM_Noed;
11
12 #define LCDM_ERREUR_MEMOIRE 1
13
14 LCDM_ListeChaine LCDM_listeChaine();
15 int LCDM_estVide(LCDM_ListeChaine);
16 void LCDM_ajouter(LCDM_ListeChaine*, M_Monome); /* errno=LCDM_ERREUR_MEMOIRE si pas assez de mémoire */
17 M_Monome LCDM_obtenirElement(LCDM_ListeChaine); /* assertion : liste non vide */
18 LCDM_ListeChaine LCDM_obtenirListeSuiivante(LCDM_ListeChaine); /* assertion : liste non vide */
19 void LCDM_fixerListeSuiivante(LCDM_ListeChaine*, LCDM_ListeChaine); /* assertion : liste non vide */
20 void LCDM_fixerElement(LCDM_ListeChaine*, M_Monome); /* assertion : liste non vide */
21 void LCDM_supprimerTete(LCDM_ListeChaine*); /* assertion : liste non vide */
22 void LCDM_supprimer(LCDM_ListeChaine*);
23 LCDM_ListeChaine LCDM_copier(LCDM_ListeChaine);
24 int LCDM_egale(LCDM_ListeChaine, LCDM_ListeChaine);
25 #endif

```

# Première proposition : Développer une liste chaînée de monomes 3 / 8

## ListeChaineDeMonomes.c

```
1 #include <stdlib.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <assert.h>
5 #include <stdbool.h>
6 #include "ListeChaineDeMonomes.h"
7
8 LCDM_ListeChaine LCDM_listeChaine(){
9     errno=0;
10    return NULL;
11 }
12
13 int LCDM_estVide(LCDM_ListeChaine l) {
14     errno=0;
15     return (l==NULL);
16 }
```

# Première proposition : Développer une liste chaînée de monomes 4 / 8

## ListeChaineDeMonomes.c

```
18 void LCDMajouter(LCDM_ListeChaine* pl, M_Monome m) {
19     LCDM_ListeChaine pNoeud=(LCDM_ListeChaine)malloc(sizeof(LCDM_Noeud));
20     if (pNoeud!=NULL) {
21         errno=0;
22         pNoeud->IElement=m;
23         pNoeud->listeSuivante=*pl;
24         *pl=pNoeud;
25     } else {
26         errno=LCDM_ERREUR_MEMOIRE;
27     }
28 }
29
30 M_Monome LCDMobtenirElement(LCDM_ListeChaine l) {
31     assert (!LCDM_estVide(l));
32     errno=0;
33     return l->IElement;
34 }
```

# Première proposition : Développer une liste chaînée de monomes 5 / 8

## ListeChaineDeMonomes.c

```
36 LCDM_ListeChainee LCDM_obtenirListeSuiivante(LCDM_ListeChainee l) {
37     assert (!LCDM_estVide(l));
38     errno=0;
39     return l->listeSuiivante;
40 }
41
42 void LCDM_fixerElement(LCDM_ListeChainee* pl, M_Monome m) {
43     assert (!LCDM_estVide(*pl));
44     errno=0;
45     (*pl)->lElement=m;
46 }
47
48 void LCDM_fixerListeSuiivante(LCDM_ListeChainee* pl, LCDM_ListeChainee suivant) {
49     assert (!LCDM_estVide(*pl));
50     errno=0;
51     (*pl)->listeSuiivante=suivant;
52 }
```

# Première proposition : Développer une liste chaînée de monomes 6 / 8

## ListeChaineDeMonomes.c

```
54 void LCDM_supprimerTete(LCDM_ListeChaine* pl){
55     LCDM_ListeChaine temp;
56     assert (!LCDM_estVide(*pl));
57     errno=0;
58     temp=*pl;
59     *pl=LCDM_obtenirListeSuiivante(*pl);
60     free (temp);
61 }
62
63 void LCDM_supprimer(LCDM_ListeChaine* pl){
64     errno=0;
65     if (!LCDM_estVide(*pl)) {
66         LCDM_supprimerTete(pl);
67         LCDM_supprimer(pl);
68     }
69 }
```

# Première proposition : Développer une liste chaînée de monomes 7 / 8

## ListeChaineDeMonomes.c

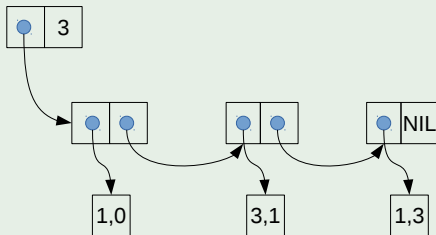
```
71 LCDM_ListeChaine LCDM_copier(LCDM_ListeChaine l) {
72     LCDM_ListeChaine temp;
73     errno=0;
74     if (LCDM_estVide(l))
75         return LCDM_listeChaine();
76     else {
77         temp=LCDM_copier(LCDM_obtenirListeSuivante(l));
78         LCDM_ajouter(&temp,LCDM_obtenirElement(l));
79         return temp;
80     }
81 }
```

# Première proposition : Développer une liste chaînée de monomes 8 / 8

## ListeChaineDeMonomes.c

```
83 int LCDM_egale(LCDM_ListeChainee l1,LCDM_ListeChainee l2) {
84     errno=0;
85     if (LCDM_estVide(l1) && LCDM_estVide(l2))
86         return true;
87     else {
88         if (LCDM_estVide(l1) || LCDM_estVide(l2))
89             return false ;
90         else {
91             if (M_egal(LCDM_obtenirElement(l1),LCDM_obtenirElement(l2)))
92                 return LCDM_egale(LCDM_obtenirListeSuiivante(l1),LCDM_obtenirListeSuiivante(l2))
93                 ;
94             else
95                 return false ;
96         }
97     }
```

## Deuxième proposition : une liste chaînée générique 1 / 9

Même exemple :  $x^3 + 3x + 1$ 

# Deuxième proposition : une liste chaînée générique 2 / 9

- Utiliser les pointeurs de fonction et pointeur générique `void*`
- Cela implique :
  - De définir les types de fonctions permettant de copier, supprimer, et comparer des éléments (`void*`) d'une collection (`copieLiberationComparaison.h`)
  - De créer la liste chaînée générique (`listeChaine.c`, `listeChaine.h`)
  - De créer les fonctions permettant de copier, supprimer, et comparer des monomes d'une collection (`monomeAllocationDynamique.h`, `monomeAllocationDynamique.c`)

Les deux premiers points sont indépendants du projet (donc réutilisable)

## copieLiberationComparaison.h

```
1 #ifndef __COPIE_LIBERATION_COMPARAISSON__
2 #define __COPIE_LIBERATION_COMPARAISSON__
3
4 typedef void* (*CLC_FonctionCopier) (void*);
5 typedef void (*CLC_FonctionLiberer) (void*);
6 typedef int (*CLC_FonctionComparer) (void*,void*);
7 #endif
```

## Deuxième proposition : une liste chaînée générique 4 / 9

## listeChaine.h

```
11 #ifndef __LISTE_CHAINEE__
12 #define __LISTE_CHAINEE__
13 #include <errno.h>
14 #include "copieLiberationComparaison.h"
15
16 /* Partie privée */
17 typedef struct LC_Noeud* LC_ListeChaine;
18 typedef struct LC_Noeud {
19     void* IElement;
20     LC_ListeChaine listeSuiivante ;
21 } LC_Noeud;
22
23 #define LC_ERREUR_MEMOIRE 1
24 LC_ListeChaine LC_listeChaine ();
25 int LC_estVide(LC_ListeChaine);
26 void LC_ajouter(LC_ListeChaine*, void*, CLC_FonctionCopier);
27 void* LC_obtenirElement(LC_ListeChaine);
28 LC_ListeChaine LC_obtenirListeSuiivante (LC_ListeChaine);
29 void LC_fixerListeSuiivante (LC_ListeChaine*, LC_ListeChaine);
30 void LC_fixerElement(LC_ListeChaine*, void*, CLC_FonctionCopier, CLC_FonctionLiberer);
31 void LC_supprimerTete(LC_ListeChaine*, CLC_FonctionLiberer);
32 void LC_supprimer(LC_ListeChaine*, CLC_FonctionLiberer);
33 LC_ListeChaine LC_copier(LC_ListeChaine, CLC_FonctionCopier);
34 int LC_egales(LC_ListeChaine, LC_ListeChaine, CLC_FonctionComparer);
```

## Deuxième proposition : une liste chaînée générique 5 / 9

## une extrait de listeChaine.c

```
18 void LC_ajouter(LC_ListeChaine* pl, void* source, CLC_FonctionCopier copierElement) {
19     LC_ListeChaine pNoeud=(LC_ListeChaine)malloc(sizeof(LC_Noeud));
20     void* donnee=copierElement(source);
21     if ((pNoeud!=NULL) && (donnee!=NULL)) {
22         errno=0;
23         pNoeud->IElement=donnee;
24         pNoeud->listeSuiivante=*pl;
25         *pl=pNoeud;
26     } else {
27         errno=LC_ERREUR_MEMOIRE;
28     }
29 }
```

## monomeAllocationDynamique.h

```
1 #ifndef __MONOME_ALLOCATION_DYNAMIQUE__
2 #define __MONOME_ALLOCATION_DYNAMIQUE__
3 #include "Monome.h"
4
5 void* MAD_copierMonome(void*);
6 void MAD_desallouerMonome(void*);
7 int MAD_comparerMonome(void*,void*);
8 #endif
```

## Deuxième proposition : une liste chaînée générique 7 / 9

## monomeAllocationDynamique.c

```
1 #include <stdlib.h>
2 #include <string.h>
3 #include "Monome.h"
4 #include "monomeAllocationDynamique.h"
5
6 void* MAD_copierMonome(void* m) {
7     M_Monome* resultat=(M_Monome*)malloc(sizeof(M_Monome));
8     memcpy((void*)resultat,m,sizeof(M_Monome));
9     return (void*) resultat ;
10 }
11
12 void MAD_desallouerMonome(void* m) {
13     free(m);
14 }
15
16 int MAD_comparerMonome(void* m1,void* m2) {
17     return M_egal(*(M_Monome*)m1,*(M_Monome*)m2);
18 }
```

## extrait de Polynome.c

```
4 P_Polynome P_polynome() {
5   P_Polynome resultat;
6   M_Monome m = M_monome(0,0);
7
8   resultat .monomes = LC_listeChaine();
9   LC_ajouter(&(resultat .monomes),&m,MAD_copierMonome);
10  resultat .degres=0;
11  return resultat ;
12 }
```

## Deuxième proposition : une liste chaînée générique 9 / 9

## extrait de Polynome.c

```
18 float P_obtenirCoefficient (P_Polynome p, unsigned int d) {
19     float resultat = 0.0;
20     LC_ListeChaine l = p.monomes;
21     int trouve = false;
22     M_Monome *monomeCourant;
23
24     while (!LC_estVide(l) && !trouve) {
25         monomeCourant = (M_Monome*)LC_obtenirElement(l);
26         if (M_degre(*monomeCourant) == d) {
27             resultat = M_coef(*monomeCourant);
28             trouve = true;
29         } else
30             l = LC_obtenirListeSuiivante(l);
31     }
32     return resultat ;
33 }
```

# Conclusion

- Le C ne propose pas par défaut la généricité
- Lorsque l'on est utilisateur d'une SDD, trois possibilités :
  - 1 Développer autant de SDD spécifiques
  - 2 Développer des SDD génériques (utilisation du `void*`)
  - 3 Utilisez des bibliothèques externes, par exemple GLib du projet GTK+, QT, etc.
  - 4 Vous trouverez sur le git du cours des implantations génériques des SDD et des implantations génériques des collections