Introduction aux méthodes Orientées Objets

Exercices

Modélisation avec UML 2.0

Programmation orientée objet en C++

Pré-requis:

maitrise des bases algorithmiques (cf. 1ier cycle),
maitrise du C (variables, fonctions, pointeurs, structures)

Thomas Grenier

Tableau dynamique **

Enoncé

```
class TableauDouble
{
double *Tab;
int Taille;
public:
  TableauDouble(int taille=1)
        :Taille(taille)
   { Tab = new double[Taille]; }
  double GetElement(int i) const
   { return Tab[i]; }
  void SetElement(int i, double v)
   { Tab[i] = v; }
  int GetTaille() const
   { return Taille; }
  ~TableauDouble()
   { delete[] Tab;}
};
```

1- Modéliser cette classe.

2- Ajouter la méthode *SetTaille.* Décrire son fonctionnement.

3- Réaliser la méthode *SetTaille.*

4- Réaliser la méthode *Copie* qui permet de faire une copie d'un objet *TableauDouble* passé en paramètre dans l'objet courant.

5- Donner un exemple d'utilisation

Tableau dynamique

Solution

```
class TableauDouble
{
double *Tab;
int Taille;
public:
  TableauDouble(int taille=1)
        :Taille(taille)
   { Tab = new double[Taille]; }
  double GetElement(int i) const
   { return Tab[i]; }
  void SetElement(int i, double v)
   { Tab[i] = v; }
  int GetTaille() const
   { return Taille; }
  ~TableauDouble()
   { delete[] Tab;}
};
```

1- Modéliser cette classe

–

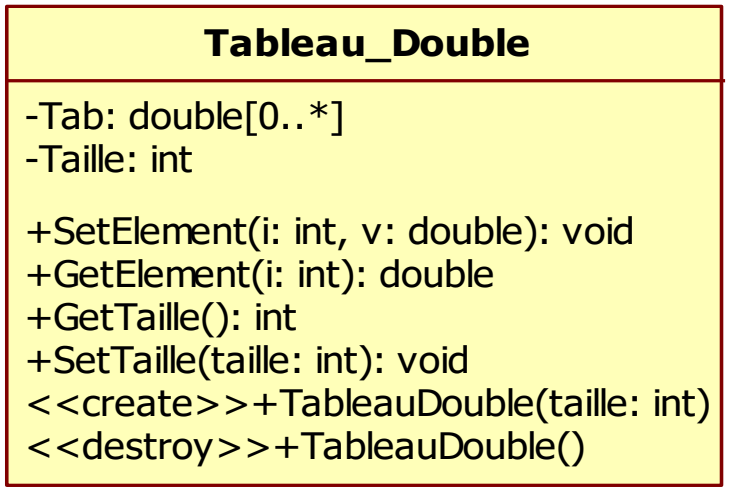
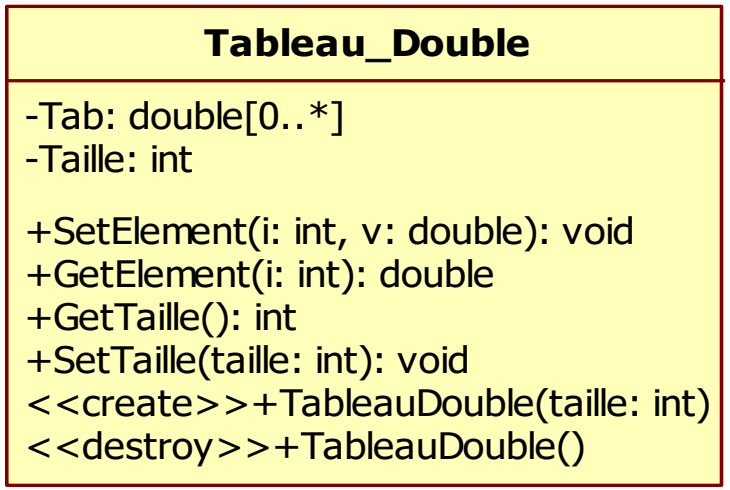


Tableau dynamique

Solution

```
class TableauDouble
{
double *Tab;
int Taille;
public:
  TableauDouble(int taille=1)
        :Taille(taille)
   { Tab = new double[Taille]; }
  double GetElement(int i) const
   { return Tab[i]; }
  void SetElement(int i, double v)
   { Tab[i] = v; }
  int GetTaille() const
   { return Taille; }
  ~TableauDouble()
   { delete[] Tab;}
  void SetTaille(int taille);
};
```

2- Ajouter la méthode *SetTaille.* Décrire son fonctionnement.



SetTaille doit:

a- supprimer l'espace alloué

b- allouer un nouvel espace

c- mettre à jour le champs Taille

Tableau dynamique

Solution

```
class TableauDouble
{
double *Tab;
int Taille;
public:
  TableauDouble(int taille=1)
         :Taille(taille)
   { Tab = new double[Taille]; }
  double GetElement(int i) const
   { return Tab[i]; }
  void SetElement(int i, double v)
   { Tab[i] = v; }
  int GetTaille() const
   { return Taille; }
  ~TableauDouble()
   { delete[] Tab;}
  void SetTaille(int taille);
};
```

3- Réaliser la méthode *SetTaille.*

SetTaille doit:

a- supprimer l'espace alloué

b- allouer un nouvel espace

c- mettre à jour le champs Taille

```
void TableauDouble::SetTaille
 (int taille)
{
if( Taille != taille )
  {
  delete[] Tab;
  Tab = new double[taille];
  Taille = taille;
  }
}
```

Tableau dynamique

Solution

```
class TableauDouble
{
double *Tab;
int Taille;
public:
  TableauDouble(int taille=1)
         :Taille(taille)
   { Tab = new double[Taille]; }
  double GetElement(int i) const
   { return Tab[i]; }
  void SetElement(int i, double v)
   { Tab[i] = v; }
  int GetTaille() const
   { return Taille; }
  ~TableauDouble()
   { delete[] Tab;}
  void SetTaille(int taille);
  void Copie( const TableauDouble &t);
};
```

4- Réaliser la méthode *Copie* qui permet de faire une copie d'un objet *TableauDouble* passé en paramètre dans l'objet courant.

a- supprimer l'espace alloué
b- allouer un nouvel espace
c- mettre à jour le champs Taille
d- copier les valeurs du tableau de l'objet passé en paramètre

```
void TableauDouble::Copie
 (const TableauDouble &t)
{
SetTaille( t.GetTaille() );
for(int i=0;i<Taille;i++)
  Tab[i] = t.GetElement(i);
}
```

Tableau dynamique

Solution

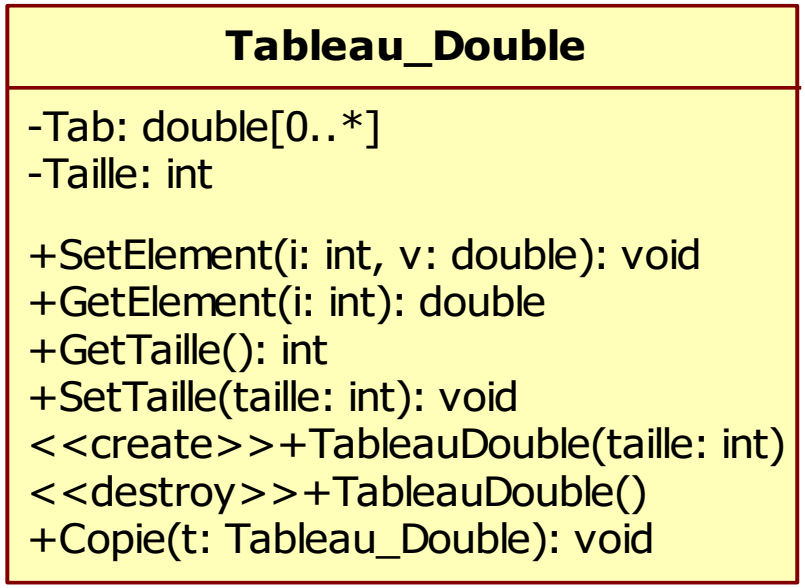
```
void main()
{
int taille;
int i;
double tmp;
TableauDouble copie;

cout << "Taille du tableau ?";
cin >> taille;
TableauDouble tab(taille);

for( i=0; i<taille; i++)
  {
  cout << "Tab[" << i << "]";
  cin >> tmp;
  tab.SetElement(i, tmp);
  }
copie.Copie( tab );
for( i=0; i<taille; i++)
  {
  cout << "Tab[" << i << "] = ";
  cout << copie.GetElement(i)<< endl;
  }
}
```

5- Donner un exemple d'utilisation

–



DeptGE *

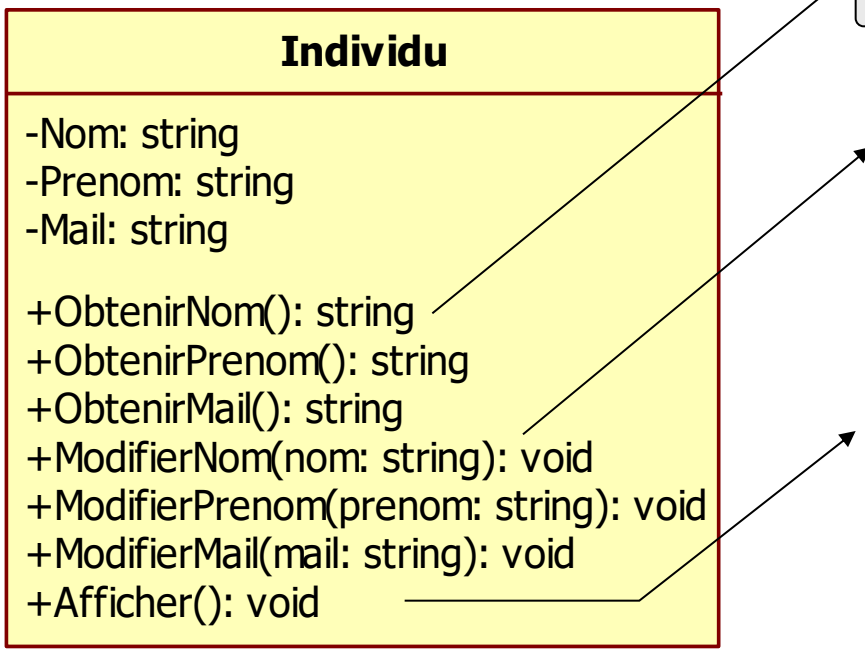
Enoncé

- Modéliser les différents effectifs du département GE. Pour cela, on modélisera les différents constituants ainsi :
  - un étudiant par: nom, prénom, promo, mail, groupe
  - un enseignant par: nom, prénom, mail, matière
  - une secrétaire par: nom, prénom, mail, fonction
  - un technicien par: nom, prénom, mail, spécialité
- On utilisera la classe Individu dont les membres sont:
  - Champs : nom, prénom et mail (de type string)
  - Fonctions d'accès en lecture et écriture à tous les champs
  - Une fonction d'affichage des 3 champs

DeptGE

Solution

- Classe Individu




```
string Individu::ObtenirNom()const
{ return Nom; }
```

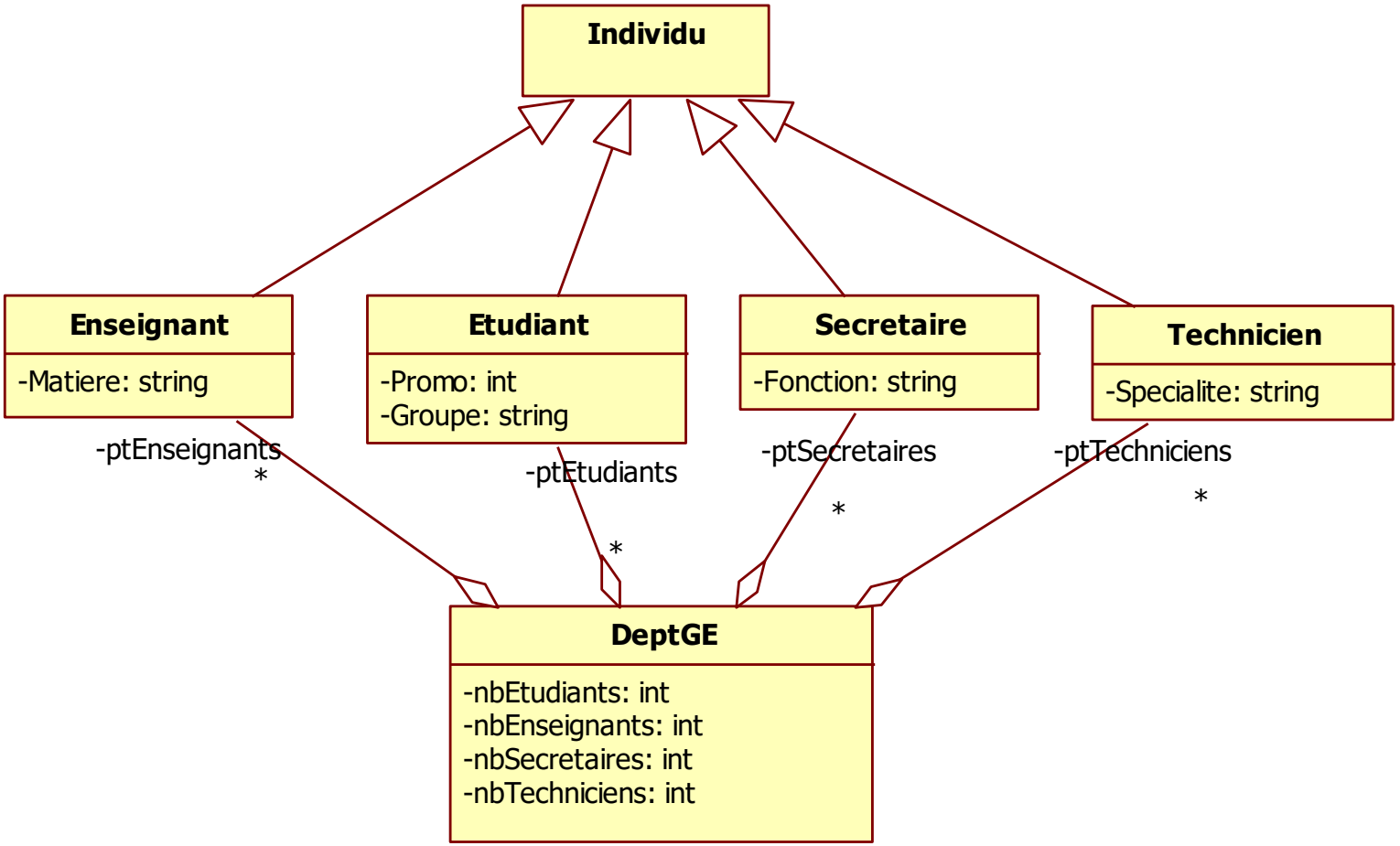
```
void Individu::ModifierNom
                        (string nom)
{ Nom = nom; }
```

```
void Individu::Afficher()const
{
cout << Nom << endl;
cout << Prenom << endl;
cout << Mail << endl;
}
```

DeptGE

Solution

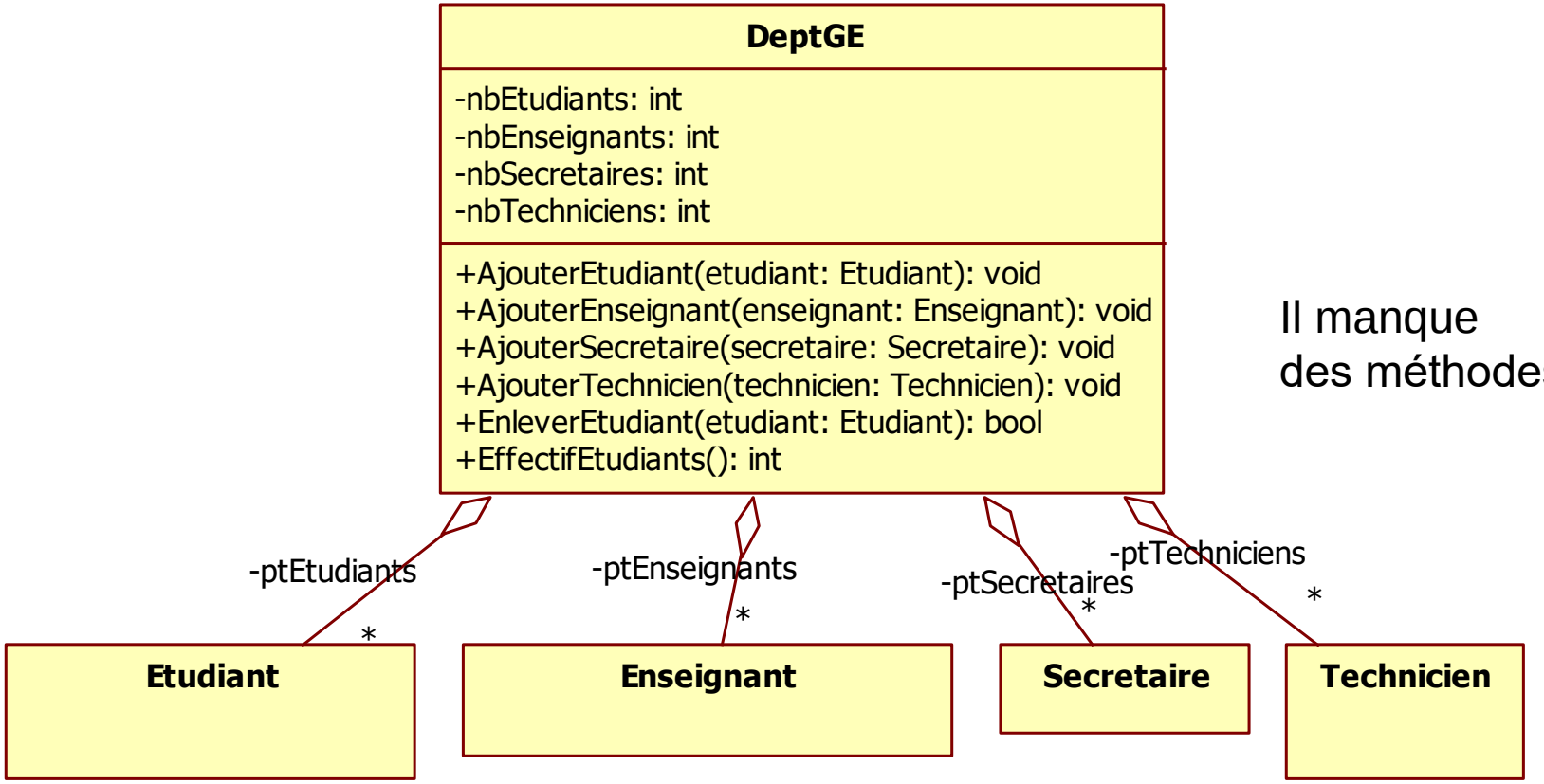
- Modélisation



DeptGE

Solution

- Modélisation



Il manque des méthodes…

DeptGE

Solution

- C++ des champs de DeptGE

```
class DeptGE
{
private:
  int nbEtudiants;
  int nbEnseignants;
  int nbSecretaires;
  int nbTechniciens;
  Enseignant **ptEnseignants;
  Etudiant   **ptEtudiants;
  Secretaire **ptSecretaires;
  Technicien **ptTechniciens;
public:
  …
  void AjouterEtudiant(Etudiants *etudiant);
  void AfficherEtudiants() const;
};
```

DeptGE

Solution

- Réalisation C++, AjouterEtudiant

```
void DeptGE::AjouterEtudiant(Etudiants *etudiant)
{
Etudiant **tmp;
tmp = new Etudiant*[ nbEtudiants+1 ];

for( int i=0;i<nbEtudiants;i++)
  tmp[i] = ptEtudiants[i];
tmp[nbEtudiants] = etudiant;

delete[] ptEtudiants;
ptEtudiants = tmp;

nbEtudiants++;
}
```

Allocation nouvel espace mémoire

Copie des adresses des anciens étudiants, puis celle du nouveau

Libération de l'ancien espace mémoire

Utilisation du nouvel espace

Mise à jour du nombre d'étudiants

DeptGE

Solution

- Réalisations C++:

  Etudiant::Afficher() et DeptGE::AfficherEtudiants()

```
void Etudiant::Afficher() const
{
  // cout << Nom << Prenom << Mail;
  Individu::Afficher();
  cout << Promo << Groupe;
}
```

Non !
Pour la classe dérivée les champs privés de la classe mère sont inaccessibles

Ok, méthode publique de la classe mère

```
void DeptGE::AfficherEtudiants() const
{
for( int i=0; i<nbEtudiants; i++)
  {
  cout << "Etudiant " << i << " : ";
  ptEtudiants[i]->Afficher();
  cout << endl;
  }
}
```

ptEtudiants est du type: Etudiant **

Exercice équipe de sport Dept ***

- A partir de la modélisation d'un département (classes Dept, Individu, Etudiant, Enseignant, …), modéliser **une équipe** dont les joueurs sont un mélange d'étudiants, d'enseignants, de secrétaires et de techniciens appartenant tous à un même département.
  - On voudra afficher la liste des joueurs d'une équipe, en précisant tous les attributs de chaque joueur (méthode afficher)
  - Pour les feuilles de match, on ne souhaite afficher que le nom et le prénom (méthode AfficherFeuilleMatch())

Article de journal ***

Enoncé

- Modéliser les différents éléments d'un article de journal. La modélisation sera focalisée sur la mise en forme du texte (Titre, SousTitre(s), Auteur(s), Illustration(s), CorpsTexte(s)). Prévoir l'accès aux différents champs et l'affichage complet de l'article.
- On dispose d'une classe *FaitActualité*. Quel lien existe-t-il entre cette classe et la classe précédente ?

Commode de rangement **

Enoncé

- Modéliser les différents éléments du système suivant:
  - une commode de 3 tiroirs permettant de ranger des vêtements (Pull, Chemise, TShirt, Chaussettes),
  - Les tiroirs ont un volume donné (20),
  - Les vêtements sont caractérisés par un nom, une couleur et un volume d'encombrement:
    - Un Pull : nom=« Pull », encombrement = 4
    - Une Chemise: nom= « Chemise », encombrement = 3
    - Un TShirt : nom=« TShirt », encombrement = 2
    - Une paire de Chaussettes : nom=« Chaussette », encombrement = 1
  - Pour chaque vêtement, on dispose d'une méthode 'Enfiler' spécifique à chaque vêtement
- On veut pouvoir Ajouter, Enlever, Chercher et Afficher le contenu d'un tiroir

Course *

Enoncé

- Il s'agit de gérer les temps et les participants d'une course. Les coureurs peuvent s'inscrire quand ils veulent et partent quand ils veulent.
- Pour gérer la course, on souhaite
  - Ajouter un participant, modifier son heure de départ et son heure d'arrivée
  - Afficher les temps de tous les participants (ordre alphabétique)
  - Savoir combien de participants sont inscrits, courent et sont arrivés
  - Savoir si la course est terminée (tout le monde est arrivé)
  - Afficher la liste des participants dans l'ordre des temps croissants

➔ Modéliser le système.

Polynôme *

Enoncé

- Modéliser la classe ***Polynome*** permettant de:
  - stocker les n+1 coefficients du polynôme d'ordre n
  - Accéder à chaque coefficient (lecture et écriture)
  - Evaluer le polynôme pour ***x*** donné (P(***x***) =…)
  - Additionner deux polynômes et retourner le nouveau polynôme
- Ecrire en C++
  - les fonctions d'accès aux coefficients
  - la fonction pour évaluer P(***x***)

$P(x)=a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \rightarrow (\ldots(a_n.x + a_{n-1}).x + \ldots + a_1)x + a_0$