Cours WinDev Numéro 10

Objectifs : Les listes d'objets doublement chaînées.

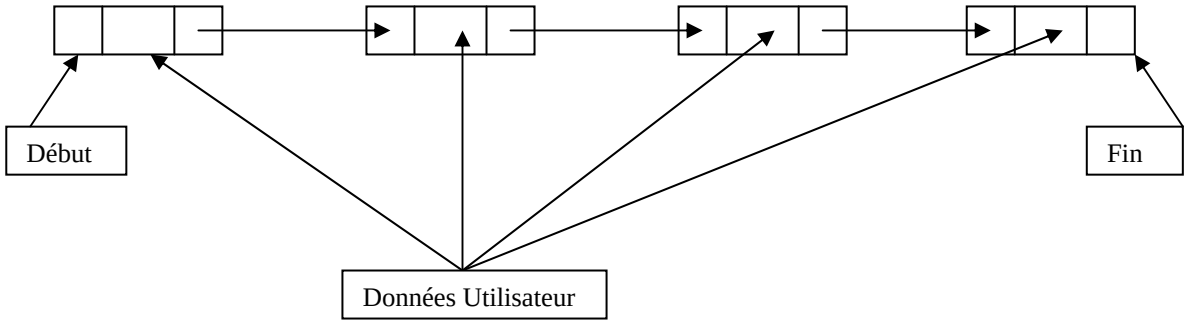
Création d'objets dynamique.
Création de liste.
Ajout dans une liste.
Suppression d'un objet dans la liste.
La récursivité.

Pré-requis : Avoir fait les cours sur les objets

Les listes chaînées constituent une alternative aux tableaux. L'ajout, l'insertion et le retrait d'information y sont très rapides et la longueur totale de la liste est inconnue au départ. Cependant, la recherche est lente, la sauvegarde délicate.

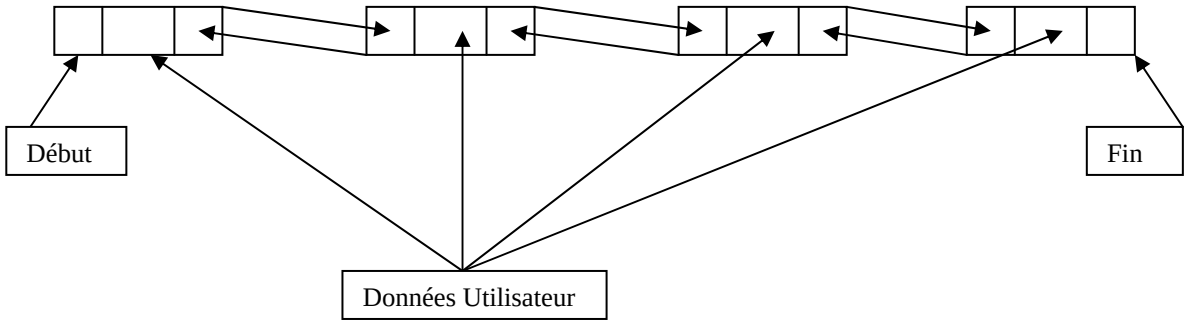
Une liste chaînée est un ensemble de cellules, d'objets, de structures, relié entre eux par des pointeurs

On pourrait représenter une liste chaînée comme ceci :



L'inconvénient majeur des listes simplement chaînées est qu'on ne peut les parcourir que du début vers la fin.

La solution c'est la liste doublement chaînée :



Ce nouveau support va vous faire programmer une liste doublement chaînée.

Pour commencer, créez un nouveau projet nommé "ListeDC", ce projet ne gère aucune analyse. Il contiendra une fenêtre et une classe.

La fenêtre :

Créez une nouvelle fenêtre que vous nommerez "Départ".

Voici ses dimensions : 350x230

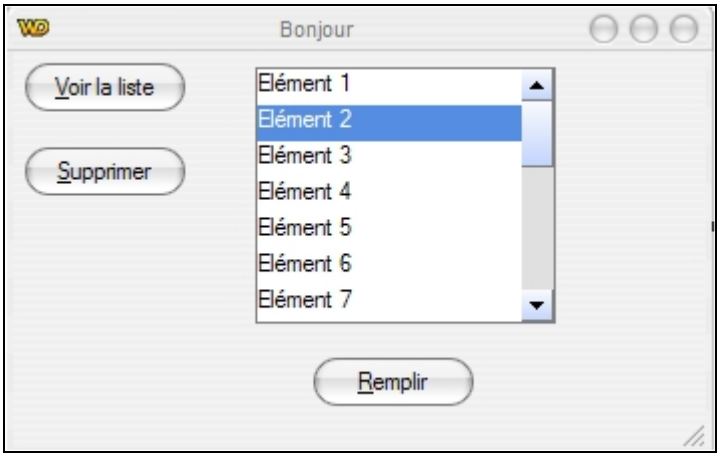


Tableau d'objets :

Type de champ	Nom	Libelle	Position relative	Taille
Bouton	Bremplir	Remplir	150, 153	80X24
Bouton	Bvoir	Voir la liste	6, 6	80X24
Bouton	Bsupprime	Supprimer	6, 48	80X24
Champ liste	Listeprénom	Aucun	121, 8	150x128

Vous n'oublierez pas de mettre cette fenêtre comme première du projet.

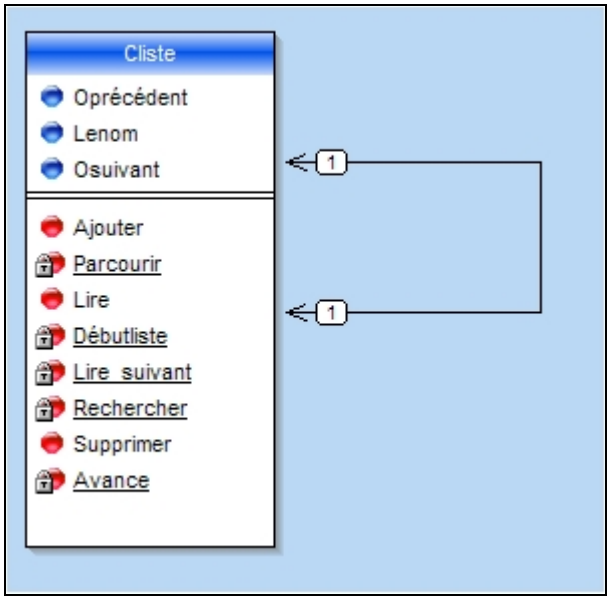
Le rôle du bouton "**Remplir**" est de remplir la liste doublement chaînée d'éléments, ici une liste de prénoms et de les ajouter dans la liste chaînée.

Le bouton "**Voir la liste**" efface les éléments présents dans la zone d'affichage "**liste**" et inscrit ceux contenus dans la liste chaînée.

Le bouton "**Supprimer**" supprime à la fois dans la liste chaînée et dans la zone d'affichage "**liste**".

Voilà pour le décor, passons à la réalisation. Pour que tout fonctionne, nous avons besoin d'une classe nommée "**Cliste**".

Je vous laisse la concevoir à partir de ce diagramme UML.



Voyons ensemble cette classe.

Les attributs comportent :

- Un objet nommé Oprécédent chargé de contenir la référence de l'objet Cliste précédent (ce qui explique le lien de ré-entrance).
- Un objet nommé Osuivant chargé de contenir l'objet Cliste suivant.
- Une chaine nommée Lenom qui contiendra la valeur à faire afficher.

Les méthodes publiques sont celles qui seront nécessaires à la mise en œuvre de la classe :

Pour faire fonctionner une liste nous avons besoin :

- D'ajouter un élément à l'intérieur,
- De supprimer un élément,
- De lire.

Nous aurions aussi besoin d'insérer et de modifier mais je vous laisse le soin de créer vous même ces 2 méthodes là !

Les méthodes privées sont des méthodes fonctionnelles inaccessibles de l'extérieur. Elles servent aux différents positionnements ou déplacement dans la liste. Pour les besoins du cours, il n'y a aucune optimisation donc les puristes vont pouvoir trouver de quoi s'amuser en exercice.

Je pense que vous venez de voir quelles méthodes vont activer les 3 boutons ?

On décortique la classe ?

```
Déclaration de Cliste
Cliste est une classe
    Oprécédent est un Cliste dynamique
    Lenom est une chaîne
    Osuivant est un Cliste dynamique
FIN

Terminaison de Cliste


Constructeur
PROCEDURE Constructeur(Pnom="")
:Lenom=pnom

Destructeur
PROCEDURE Destructeur()
```

Comme vous le voyez dans la déclaration de la classe Cliste on lui signifie qu'elle contiendra 2 objets Cliste dynamiques (par référence).

La méthode ajouter :

```
Méthode Ajouter
PROCEDURE Ajouter(Pnom)
SI pnom<>"" ALORS
    SI :Lenom<> "" ALORS
        ::Parcourir(objet,pnom)
    SINON
        :Lenom=Pnom
    FIN
FIN
```

Explications : La méthode ajouter est chargée d'ajouter une valeur dans notre liste chaînée. On vérifie que la valeur passée en paramètre est valide. Ensuite on teste si l'attribut "**lenom**" de l'objet en cours est affecté ou pas s'il n'est pas affecté on lui affecte la variable passée en paramètre sinon on appelle la méthode parcourir en lui passant en paramètre l'objet en cours et la valeur à ajouter dans la liste chaînée.

La méthode Parcourir :

```
Méthode Parcourir
PROCEDURE PRIVÉE GLOBALE Parcourir(Pobjet,Pvaleur)

SI Pobjet:Osuivant=Null ALORS
    Pobjet:Osuivant=allouer un Cliste(Pvaleur)
    pobjet:Osuivant:Oprécédent=Pobjet
SINON
    ::Parcourir(Pobjet:Osuivant,Pvaleur)
FIN

Méthode Lire
```

La méthode parcourir à comme but de parcourir la liste chaînée vers la fin jusqu'à ce qu'elle trouve un emplacement "Objet suivant" vide (Null). Dès qu'elle a une place libre, elle y affecte la référence du nouvel objet.
Toute l'élégance de cette méthode tient au fait quelle est récursive, c'est à dire quelle se rappelle jusqu'à ce quelle trouve une place libre. Remarquez la chose suivante : au début la méthode Parcourir a comme paramètre objet l'objet en cours, ensuite en récursivité elle prend l'objet suivant et si elle n'a pas trouvé, elle progresse d'objet suivant en objet suivant.
Je suis clair ?

Voici une autre méthode utilisant la récursivité. Son but est de renvoyer le premier objet de la liste chaînée. Etudiez son comportement.

La méthode Débutliste :

Méthode **Débutliste**

```
PROCEDURE PRIVÉ GLOBALE Débutliste(Pobjet)
 Lobjet est un Cliste dynamique

SI Pobjet:Oprécédent<>Null ALORS
     lobjet=::Débutliste(pobjet:Oprécédent)
FIN
 lobjet=Pobjet
 RENVOYER lobjet
```

La méthode lire:

Cette méthode fait passer l'objet en cours à la méthode "**débutliste**" qui recherche le premier objet de la liste chaînée et donne la référence du premier objet à "**lobjet**". Une fois ceci réalisé, la méthode supprime tout le contenu de la liste de la fenêtre "**départ**". Ensuite, elle enclenche une méthode de lecture de la liste chaînée.

Méthode **Lire**

```
PROCEDURE Lire()
 lobjet est un Cliste dynamique=::Débutliste(objet)
 ListeSupprimeTout(Départ.ListePrénom)
 ::Lire_suivant(lobjet)
```

Méthode Lire_suivant:

Cette méthode récursive remplit la liste de la fenêtre "**départ**" du début de la liste chaînée à la fin. Je vous laisse l'analyser.

Méthode **Lire_suivant**

```
PROCEDURE PRIVÉ GLOBALE Lire_suivant(Pobjet)
 Lobjet est un Cliste dynamique

SI Pobjet:Osuivant<>Null ALORS
     ListeAjoute(Départ.ListePrénom,Pobjet:Lenom)
     lobjet=::Lire_suivant(pobjet:Osuivant)
 SINON
     ListeAjoute(Départ.ListePrénom,Pobjet:Lenom)
FIN
 Lobjet=Pobjet
 RENVOYER lobjet
```

La méthode Rechercher :

Comme vous le voyez cette méthode recherche dans un objet une valeur, pour progresser dans la liste chaînée il emploie la méthode "**Avance**".

Méthode **Rechercher**

```
PROCEDURE PRIVÉ GLOBALE Rechercher(pobjet,pvaleur)
 Lobjet est un Cliste dynamique

SI Pobjet:Lenom<>pvaleur ALORS
     lobjet=::Avance(pobjet:Osuivant,pvaleur)
FIN
 RENVOYER lobjet
```

La méthode Avance :
C'est beau la récursivité, je ne m'en lasse pas…Allez je vous laisse comprendre ce qu'elle fait, c'est simple.

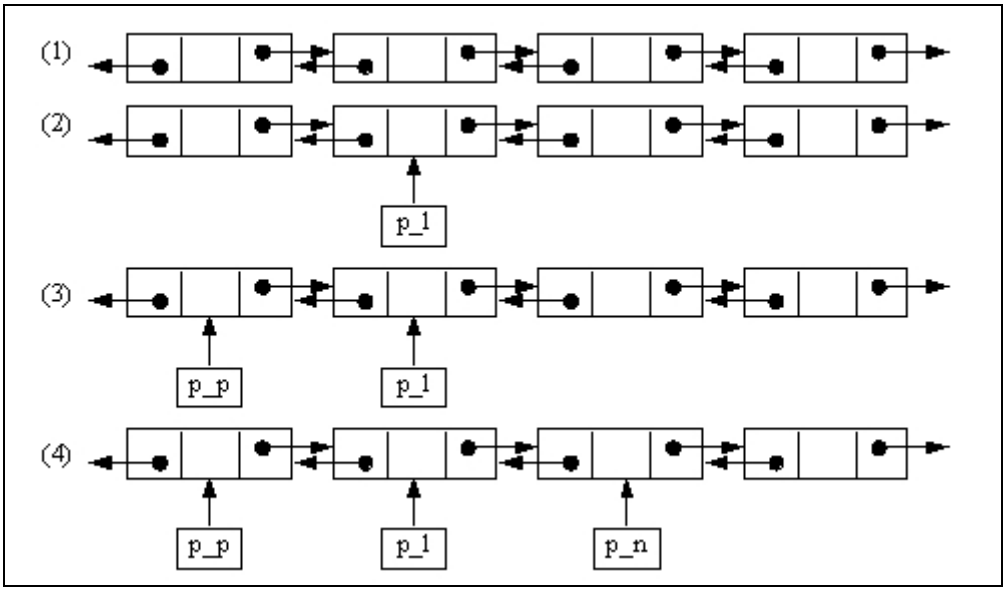
Méthode **Avance**

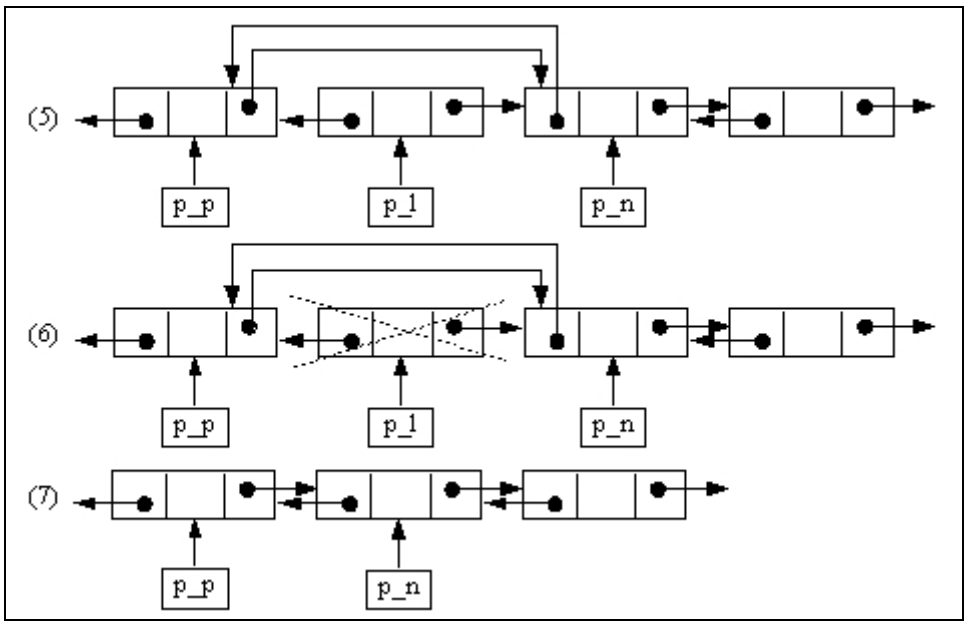
```
PROCEDURE PRIVÉ GLOBALE Avance(pobjet,pvaleur)
 Lobjet est un Cliste dynamique

SI Pobjet:Osuivant<>Null ET pobjet:Lenom<>pvaleur ALORS
     lobjet=::Avance(pobjet:Osuivant,pvaleur)
SINON
     SI pobjet:Lenom=pvaleur ALORS
         Lobjet=Pobjet
     FIN
FIN
 RENVOYER lobjet
```

La méthode Supprimer :

Celle là c'est ma préférée ! Elle utilise la méthode "**débutliste**" pour se positionner en début de liste, elle lance ensuite la méthode de recherche de valeur.
Une fois l'objet contenant la valeur trouvé, elle le supprime. En fait elle ne le supprime pas pour de vrai, on pourrait dire qu'elle le débranche de la liste. Regardez ce qu'elle fait sur ce dessin (on débranche p_1) :





C'est magique, non ? Voici le code :

Méthode **Supprimer**

```
PROCEDURE Supprimer(pnom)
lobjet est un Cliste dynamique=::Débutliste(objet)
lobjetsup est un Cliste dynamique=::Rechercher(lobjet,pnom)

SI lobjetsup:Oprécédent<>Null ALORS
    lobjetsup:Oprécédent:Osuivant=lobjetsup:Osuivant
FIN
SI lobjetsup:Osuivant<>Null ALORS
    lobjetsup:Osuivant:Oprécédent=lobjetsup:Oprécédent
FIN
```

Voilà la définition de la classe est finie.
Voyons le code des objets de la fenêtre :

On démarre par le code de la fenêtre "**départ**" :
Ici on définit un objet "**maliste**" de type "**Cliste**"

Déclarations globales de **Départ**

```
maliste est un Cliste
```

Voyons maintenant le code du bouton **"Bremplir"**.
Comme vous le voyez, il vide la liste contenant les prénoms, ensuite il la remplit avec les valeurs de bases et enfin il ajoute tout les éléments de la liste dans la liste doublement chaînée.

```
Clic sur Bremplir
i est un entier

ListeSupprimeTout(ListePrénom)

ListeAjoute(ListePrénom,"Béatrice")
ListeAjoute(ListePrénom,"Amandine")
ListeAjoute(ListePrénom,"Cédric")
ListeAjoute(ListePrénom,"Sylvain")
ListeAjoute(ListePrénom,"Jean-Luc")
ListeAjoute(ListePrénom,"Baptiste")


POUR i=1 A ListeOccurrence(ListePrénom)
    maliste:Ajouter(ListePrénom[i])
FIN

Info("La liste chainée est remplie")
```

Voici le code du bouton **"Bsupprime"** :

On supprime la valeur pointée et ensuite on fait réafficher la liste doublement chaînée.

```
Clic sur Bsupprime
maliste:Supprimer(ListePrénom..ValeurAffichée)
maliste:Lire()
```

Voici le dernier code, celui du bouton "**Bvoirliste**" :

```
Clic sur Bvoirliste
maliste:Lire()
```

On exécute quelques clics ?
Lancer le projet et commencez par cliquer sur le bouton voir liste, comme vous pouvez le constater rien ne s'affiche dans la zone liste.
Maintenant cliquez sur remplir, la liste de prénom est remplie ainsi que la liste doublement chaînée.
Sélectionnez un élément (moi par exemple NdL : moi c'est Jean-Luc) et hop on le supprime !
Magique, je viens de disparaître !

Voilà, je viens de finir de vous parler des listes doublement chaînées.

Exercices.

Pour vous entraîner :

- Cliquez sur le premier élément de la liste : Comme vous le voyez il y a un big problème. Je vous laisse analyser le problème et y trouver une solution (en fait, il y en a deux possibles)

Programmez les méthodes permettant :

- D'insérer un élément avant la position de l'élément sélectionné.
- De trier la liste par ordre croissant ou décroissant.