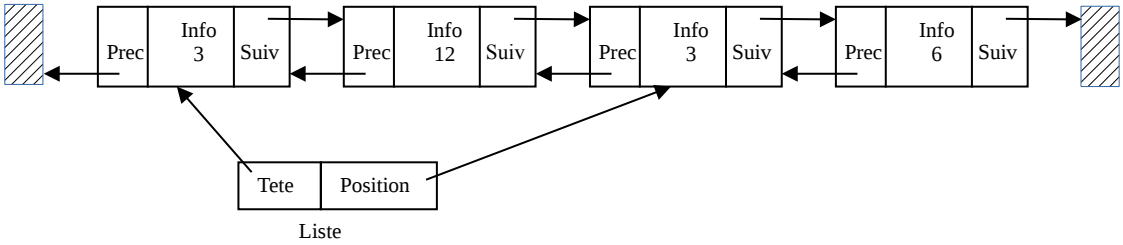
Algorithmique & Structures de données

Listes doublement chaînées

Structure de liste doublement chaînée

On considère une liste doublement chaînée d'articles. Chaque article comportera un champ pour stocker les valeurs dans la liste et deux champs contenant une variable de type accès pour assurer le chaînage vers les éléments suivant et précédent.

Schématiquement:



L'accès à la liste se fait par les pointeurs `Tete` et `Position` qui référencent respectivement le début de la liste et une position courante dans la liste.

Une telle liste doublement chaînée sera déclarée par exemple sous la forme suivante:

```
type T_Element;                     -- Pré-déclaration indispensable !
type T_Lien is access T_Element;    -- Pointeur sur des éléments de liste
type T_Element is record            -- Elément de liste
   Info : Integer ;
   Suiv : T_Lien := null;
   Prec : T_Lien := null;
end record;
type T_Liste is record
   Tete     : T_Lien := null;
   Position : T_Lien := null;
end record;
```

On peut aussi considérer un ensemble d'exceptions :

- `LISTE_VIDE` : la liste est vide
- `POSITION_INVALIDE` : le pointeur de position de la liste est non valide
- `INFO_ABSENTE` : l'information n'est pas dans la liste
- `FIN_LISTE` : le pointeur de position de la liste est en fin de liste
- `DEBUT_LISTE` : le pointeur de position de la liste est en début de liste

Quelques fonctions de consultation et procédures de manipulation

La procédure `Valider` est utilitaire et permet de vérifier que la liste est prête à l'utilisation.

```
procedure Valider(Liste : in T_Liste) is
begin
   if Est_Vide(Liste) then
      raise LISTE_VIDE;
   elsif Liste.Position = null then
      raise POSITION_INVALIDE;
   end if;
end Valider;
```

La fonction `Valeur` retourne la valeur de l'information dans l'élément pointé par le pointeur de position de la liste.

```
function Valeur(Liste : T_Liste) return Integer is
begin
   Valider(Liste);
   return Liste.Position.Info;
end Valeur;
```

La fonction `Est_Vide` indique si la liste est vide.

```
function Est_Vide(Liste : T_Liste) return Boolean is
begin
   return Liste.Tete = null;
end Est_Vide;
```

La fonction `Est_Premier` indique si le pointeur de position de la liste pointe sur le premier élément.

```
function Est_Premier(Liste : T_Liste) return Boolean is
begin
   Valider(Liste);
   return Liste.Position.Prec = null;
end Est_Premier;
```

La fonction `Est_Dernier` indique si le pointeur de position de la liste pointe sur le dernier élément.

```
function Est_Dernier(Liste : T_Liste) return Boolean is
begin
   Valider(Liste);
   return Liste.Position.Suiv = null;
end Est_Dernier;
```

La procédure `Premier` place le pointeur de position de la liste en début de liste.

```
procedure Premier(Liste : in out T_Liste) is
begin
   Valider(Liste);
   Liste.Position := Liste.Tete;
end Premier;
```

La procédure `Avancer` place le pointeur de position de la liste sur l'élément suivant de la liste et lève l'exception `FIN_LISTE` si on est déjà en fin de liste.

```
procedure Avancer(Liste : in out T_Liste) is
begin
   Valider(Liste);
   if Liste.Position.Suiv = null then
      raise FIN_LISTE;
   end if;
   Liste.Position := Liste.Position.Suiv;
end Avancer;
```

La procédure `Trouver` place le pointeur de position de la liste sur le premier élément contenant l'information recherchée et lève l'exception `INFO_ABSENTE` si celle-ci ne s'y trouve pas.

```
procedure Trouver(Liste : in out T_Liste; Info : in Integer) is
begin
   Premier(Liste);
   while Valeur(Liste) /= Info loop
      Avancer(Liste);
   end loop;
exception
   when FIN_LISTE => raise INFO_ABSENTE;
end Trouver;
```

Insertion dans une liste doublement chaînée

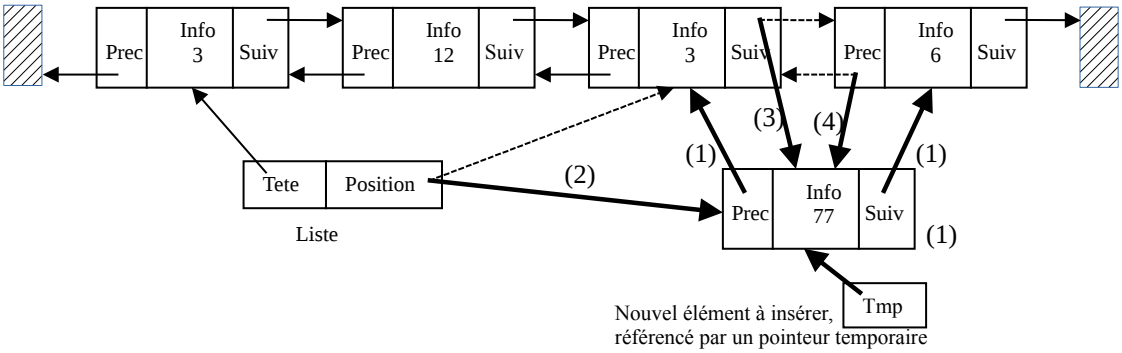
L'insertion dans la liste se fait après l'élément pointée par le pointeur de position de la liste; à la fin de la procédure, le pointeur de position pointe sur l'élément inséré.

Voici l'entête de la procédure :

```
procedure Inserer(Liste : in out T_Liste; Val : in Integer);
```

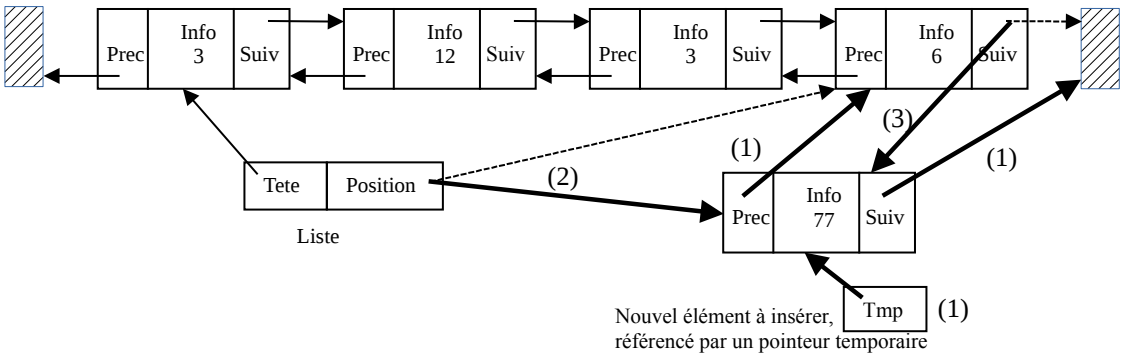
Il faut considérer plusieurs cas où on insère par exemple la valeur `Val = 77`:

a) La liste n'est pas vide et le pointeur de position n'est pas sur le dernier élément.




```
(1) Tmp := new T_Element'(77,
                          Liste.Position,
                          Liste.Position.Suiv);
(2) Liste.Position := Tmp;
(3) Tmp.Prec.Suiv := Tmp;
(4) Tmp.Suiv.Prec := Tmp;
```

b) La liste n'est pas vide et le pointeur de position est sur le dernier élément.

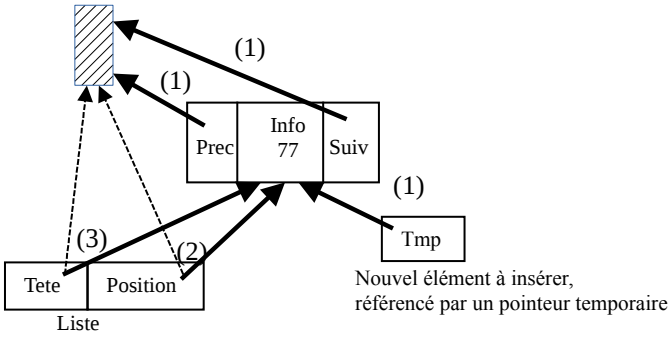



```
(1) Tmp := new T_Element'(77,
                          Liste.Position,
                          Liste.Position.Suiv);
(2) Liste.Position := Tmp;
(3) Tmp.Prec.Suiv := Tmp;
```

Les cas a) et b) peuvent être regroupés en modifiant l'instruction (4) :

```
if Tmp.Suiv /= null then
   Tmp.Suiv.Prec := Tmp;
end if;
```

c) La liste est vide.




```
(1) Tmp := new T_Element'(77,null,null);
(2) Liste.Position := Tmp;
(3) Liste.Tete := Tmp;
```

Extraction d'un élément dans une liste doublement chaînée

L'élément extrait de la liste est celui pointé par le pointeur de position de la liste; à la fin de la procédure, le pointeur de position pointe sur l'élément qui précède l'élément extrait, sauf si on extrait le premier élément.

Voici l'entête de la procédure :

```
procedure Extraire(Liste : in out T_Liste; Val : out Integer);
```

Si la liste est vide, on lève l'exception `LISTE_VIDE` via un appel : `raise LISTE_VIDE;`

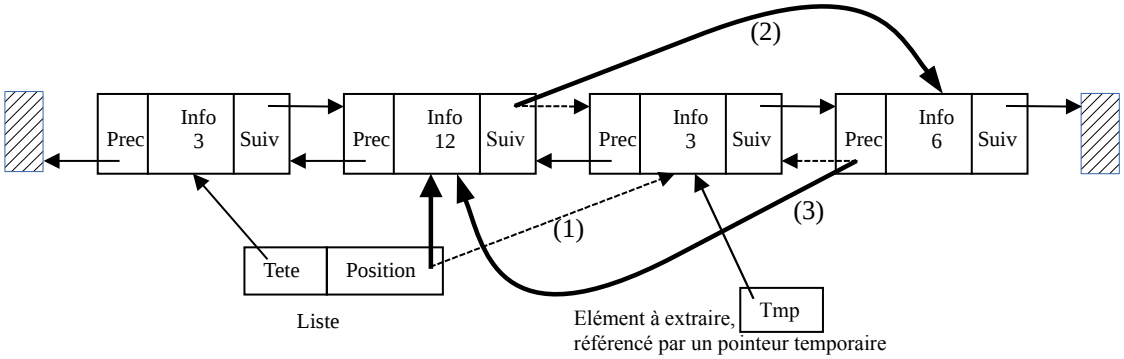
Initialement, on récupère la valeur se trouvant dans l'élément pointé par `Liste.Position`. Puis, on place un pointeur temporaire `Tmp` sur l'élément à extraire :

```
Val := Liste.Position.Info;
Tmp := Liste.Position;
```

Il faut ensuite à nouveau considérer plusieurs cas :

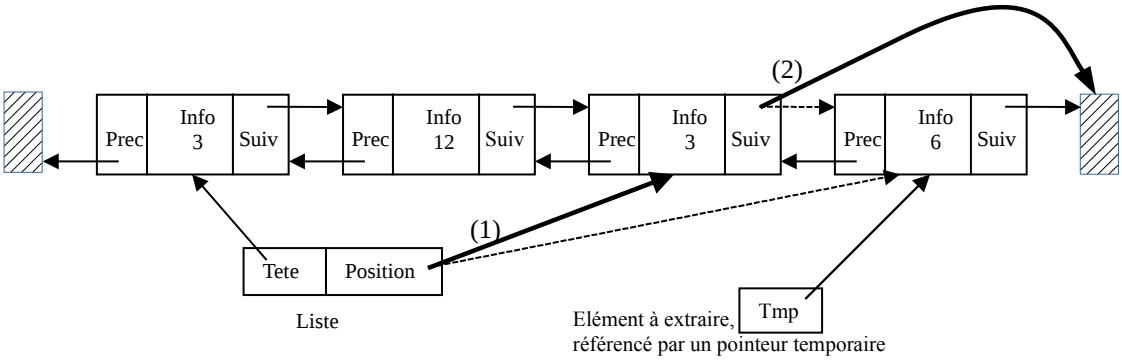
a) La liste contient plusieurs éléments.

1. On extrait un élément de milieu de liste.




```
(1) Liste.Position := Liste.Position.Prec;
(2) Tmp.Prec.Suiv := Tmp.Suiv;
(3) Tmp.Suiv.Prec := Tmp.Prec;
```

2. On extrait l'élément en fin de liste.

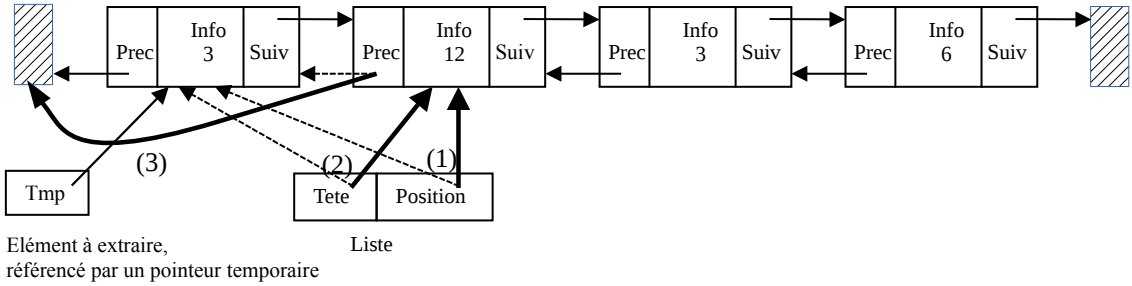



```
(1) Liste.Position := Liste.Position.Prec;
(2) Tmp.Prec.Suiv := Tmp.Suiv;
```

Les cas 1. et 2. peuvent être regroupés en modifiant l'instruction (3) :

```
if Tmp.Suiv /= null then
   Tmp.Suiv.Prec := Tmp.Prec;
end if;
```

3. On extrait l'élément de début de liste.




```
(1) Liste.Position := Liste.Position.Suiv;
(2) Liste.Tete := Liste.Position;
(3) Tmp.Suiv.Prec := Tmp.Prec;
```

b) Si la liste ne contient qu'un seul élément, il faut juste mettre les pointeurs de tête et de position de la liste à null.

```
Liste.Position := null;
Liste.Tete := null;
```

Pour rappel, la procédure `Liberer` est obtenue par instanciation via le paquetage `Ada.Unchecked_Deallocation` :

```
procedure Liberer
      is new Ada.Unchecked_Deallocation(T_Element,T_Liste);
```

Finalement, on libère la mémoire de l'élément pointé par `Tmp`:

```
Liberer(Tmp);
```

Exercice

En utilisant ce qui a été décrit dans ce chapitre, écrire un paquetage générique nommé `chaine_double` (c.-à-d. écrire les fichiers : `chaine_double.ads`, `chaine_double.adb`). Dans ce paquetage, inclure les sous-programmes suivants en plus de ceux vus précédemment :

```
function Appartient(Liste : T_Liste; Info : T_Info) return Boolean;
procedure Vider_Liste(Liste : in out T_Liste);
procedure Reculer(Liste : in out T_Liste);
```