

Plan

- Algorithme
- Types abstraits de données
- Pointeurs
- **Listes**
- Piles
- Files
- Tris
- Complexité

20/09/2006

1

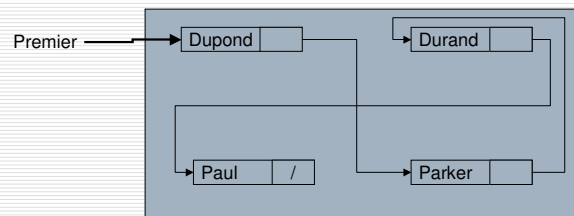
Listes > Définition

- Une liste est un ensemble d'objets de même type constituant les éléments de la liste. Les éléments sont chaînés entre eux et on peut facilement ajouter ou extraire un ou plusieurs éléments.
- Une liste simple est une structure de données telle que chaque élément contient :
 - Des informations caractéristiques de l'application développée.
 - Un pointeur vers un autre élément de la liste ou une marqueur indiquant la fin de la liste.

20/09/2006

2

Listes > Exemple



20/09/2006

3

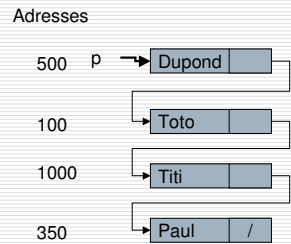
Listes > Opérations

- Créer_liste : \Rightarrow Liste
- Accès : Liste \otimes Entier \Rightarrow Place
- Contenu : Place \Rightarrow Element
- Longueur : Liste \Rightarrow Entier
- Supprimer : Liste \otimes Entier \Rightarrow Liste
- Insérer : Liste \otimes Entier \otimes Element \Rightarrow Liste
- Succ : Place \Rightarrow Place

20/09/2006

4

Listes > Représentation en mémoire > Allocation dynamique



20/09/2006

5

Listes > Représentation en mémoire > Allocation contiguë

Position	Element		Suivant
P → 0	Dupond		2
1	Paul		/
2	Toto		3
3	Titi		1
4			
5			

20/09/2006

6

Listes > Comparaison

- Allocation dynamique
 - Plus de souplesse, virtuellement pas de limite de taille
 - Gestion du dépassement de capacité à prévoir
- Allocation contiguë
 - Taille maximale doit être connue d'avance
 - Structure semblable à un fichier sur un disque

20/09/2006

7

Listes > Algorithme avec allocation dynamique > Structure de données

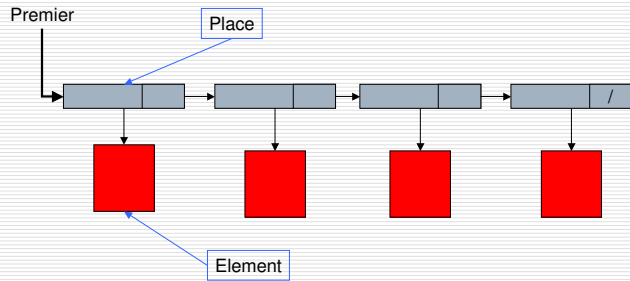
```
Enregistrement Place {
    Elt : Pointeur[Element];
    Suivant : Pointeur[Place];
}

Enregistrement Liste {
    Premier : Pointeur[Place];
}
```

20/09/2006

8

Listes > Algorithme avec allocation dynamique > Représentation en mémoire



20/09/2006

9

Listes > Algorithme avec allocation dynamique > Créer_liste



20/09/2006

10

Listes > Algorithme avec allocation dynamique > Créer_liste

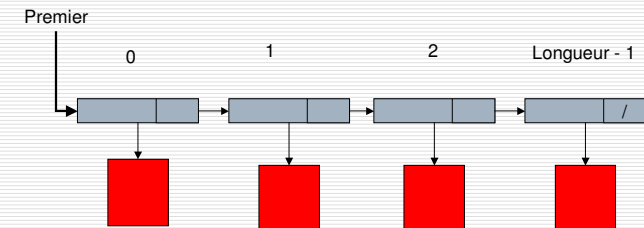
□ Créer_liste : ⇒ Liste

```
Fonction créer_liste() : Liste{  
    Resultat : Liste;  
    Resultat→Premier = null;  
    Retourne Resultat;  
}
```

20/09/2006

11

Listes > Algorithme avec allocation dynamique > Accès



20/09/2006

12

Listes > Algorithme avec allocation dynamique > Accès

- Accès : Liste @ Entier ⇒ Place

```
Fonction Accès(L : Liste; I : Entier) : Pointeur[Place]{
  Résultat : Pointeur[Place];
  Courant : Pointeur[Place];
  Compteur : Entier;
  Si (I >= 0) {
    Courant = L ⇒ Premier;
  }
  Sinon {
    Courant = null;
  } FinSi
  Compteur = 0;
  TantQue (Courant != null ET Compteur < I){
    Courant = Valeur[Courant] ⇒ Suivant;
    Compteur = Compteur + 1;
  }
  FinTantQue
  Résultat = Courant;
  Retourne Résultat;
}
```

20/09/2006

13

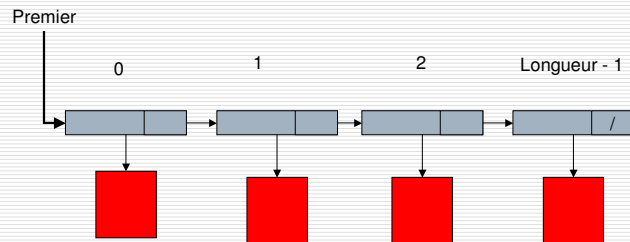
Listes > Algorithme avec allocation dynamique > Accès > Trace

- Accès(Liste_1, 6) ?
- Accès(Liste_1, -1) ?

20/09/2006

14

Listes > Algorithme avec allocation dynamique > Contenu & Longueur



20/09/2006

15

Listes > Algorithme avec allocation dynamique > Contenu & Longueur

- Contenu : Place ⇒ Element

```
Fonction Contenu (P : Place) : Pointeur[Element] {
  Retourne P ⇒ Elt;
}
```

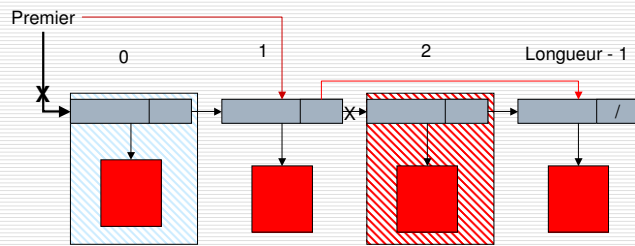
- Longueur : Liste ⇒ Entier

```
Fonction Longueur(L : Liste) : Entier {
  Taille : Entier;
  Courant : Pointeur[Place];
  Taille = 0;
  Courant = L ⇒ Premier;
  TantQue (Courant != null) Faire {
    Taille = Taille + 1;
    Courant = Valeur[Courant] ⇒ Suivant;
  }
  FinTantQue;
  Retourne Taille;
}
```

20/09/2006

16

Listes > Algorithme avec allocation dynamique > Supprimer



20/09/2006

17

Listes > Algorithme avec allocation dynamique > Supprimer

■ Supprimer : Liste \otimes Entier \Rightarrow Liste

```

Fonction Supprimer (L : Liste; Pos : Entier)
: Liste;{
Compteur : Entier;
Prec : Pointeur[Place];
Courant : Pointeur[Place];

```

```

Compteur = 1;

```

20/09/2006

18

Listes > Algorithme avec allocation dynamique > Supprimer

```

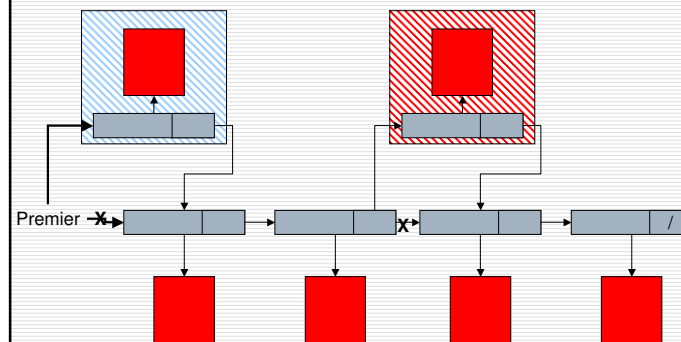
Si (L=>Premier != null) Alors {
  Si (Pos == 0) Alors {
    L=>Premier = Valeur(L=>Premier)->Suisvant;
  }
  Sinon {
    Courant = Valeur(L=>Premier)->Suisvant;
    Prec = L=>Premier;
    TantQue (Courant != null) ET (Compteur < Pos) Faire {
      Prec = Courant;
      Courant = Valeur(Courant)->Suisvant;
      Compteur = Compteur + 1;
    }
    FinTantQue
    Si (Courant != null) Et (Compteur == Pos) Alors {
      Valeur(Prec)->Suisvant = Valeur(Courant)->Suisvant;
    }
    FinSi
  }
  FinSi
}
FinSi
Retourne L;
}

```

20/09/2006

19

Listes > Algorithme avec allocation dynamique > Insérer



20/09/2006

20

Listes > Algorithme avec allocation dynamique > Insérer

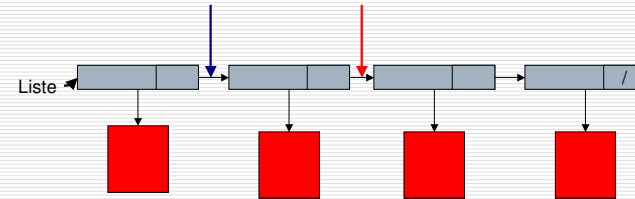
■ Insérer : Liste \otimes Entier \otimes Element \Rightarrow Liste

Fonction Insérer (L : Liste; Pos : Entier; Elt : Element) : Liste; { ...? }

20/09/2006

21

Listes > Algorithme avec allocation dynamique > Succ



20/09/2006

22

Listes > Algorithme avec allocation dynamique > Succ

■ Succ : Place \Rightarrow Place

```
Fonction Succ(P : Pointeur[Place]) :  
  pointeur[place]; {  
  Resultat : Pointeur[place];  
  Si P != null Alors  
    Resultat = Valeur[P]  $\Rightarrow$  Suivant;  
  Sinon  
    Resultat = null;  
  FinSi  
  Retourne Resultat;  
}
```

20/09/2006

23

Listes > Algorithme avec allocation dynamique > Succ > Trace

■ Succ(&P_1) ?
■ Succ(&P_4) ?

20/09/2006

24

Listes > Algorithme avec allocation contiguë > Structure de données

Constante vide : entier = -1;
Constante longueur : entier = N;

```
Enregistrement Place{  
  Elt : Element;  
  Suivant : Entier;  
  Libre : Booleen;  
}
```

```
Enregistrement Liste {  
  Premier : Entier;  
  Places : Tableau [0..longueur-1] de Place;  
}
```

20/09/2006

25

Listes > Algorithme avec allocation contiguë > Exemple

Premier → 0
Longueur → 6

Position	Elt	Libre	Suivant
0	Dupond	Faux	2
1	Paul	Faux	Vide
2	Toto	Faux	3
3	Titi	Faux	1
4	Indéfini	Vrai	Indéfini
5	Indéfini	Vrai	Indéfini

20/09/2006

26

Listes > Algorithme avec allocation contiguë > Créer_liste

Premier → Vide
Longueur → 6

Position	Elt	Libre	Suivant
0	Indéfini	Vrai	Vide
1	Indéfini	Vrai	Vide
2	Indéfini	Vrai	Vide
3	Indéfini	Vrai	Vide
4	Indéfini	Vrai	Vide
5	Indéfini	Vrai	Vide

20/09/2006

27

Listes > Algorithme avec allocation contiguë > Créer_liste

- Créer_liste : ⇒ Liste
Fonction Créer_liste : Liste {...?}

20/09/2006

28

Listes > Algorithme avec allocation contiguë > Accès

Premier → 0
Longueur → 6

Position	Elt	Libre	Suivant
0	Dupond	Faux	2
1	Paul	Faux	Vide
2	Toto	Faux	3
3	Titi	Faux	1
4	Indéfini	Vrai	Indéfini
5	Indéfini	Vrai	Indéfini

20/09/2006

29

Listes > Algorithme avec allocation contiguë > Accès

■ Accès : Liste ⊗ Entier ⇒ Place

```
Fonction accès(L : Liste; Pos : Entier) : Place;{  
  Resultat : Place;  
  Courant : Entier;  
  Compteur : entier;  
  
  Compteur = 0;  
  Courant = L⇒Premier;
```

20/09/2006

30

Listes > Algorithme avec allocation contiguë > Accès

```
TantQue (Courant != Vide) Et (Compteur < Pos) Faire {  
  Courant = L⇒Places[Courant]⇒Suivant;  
  Compteur = Compteur + 1;  
}  
FinTantQue;  
  
Si (courant != Vide) Alors {  
  Resultat = L⇒Places[Courant];  
}  
Sinon {  
  Resultat⇒Libre = Vrai;  
}  
FinSi  
Retourne Resultat;  
}
```

20/09/2006

31

Listes > Algorithme avec allocation contiguë > Accès > Trace

- Acces(Liste_2, 1) ?
- Acces(Liste_2, 5) ?

20/09/2006

32

Listes > Algorithme avec allocation contiguë > Contenu

Premier → 0
Longueur → 6

Position	Elt	Libre	Suivant
0	Dupond	Faux	2
1	Paul	Faux	Vide
2	Toto	Faux	3
3	Titi	Faux	1
4	Indéfini	Vrai	Indéfini
5	Indéfini	Vrai	Indéfini

20/09/2006

33

Listes > Algorithme avec allocation contiguë > Contenu

- Contenu : Place ⇒ Element

```
Fonction Contenu(P : Place) : Element; {  
    Retourne P⇒Elt;  
}
```

20/09/2006

34

Listes > Algorithme avec allocation contiguë > Longueur

Premier → 0
Longueur → 6

Position	Elt	Libre	Suivant
0	Dupond	Faux	2
1	Paul	Faux	Vide
2	Toto	Faux	3
3	Titi	Faux	1
4	Indéfini	Vrai	Indéfini
5	Indéfini	Vrai	Indéfini

20/09/2006

35

Listes > Algorithme avec allocation contiguë > Longueur

- Longueur : Liste ⇒ Entier

```
Fonction Longueur(L : Liste) : entier; {  
    Courant : Entier;  
    Compteur : entier;  
  
    Compteur = 0;  
    Courant = L⇒Premier;  
  
    TantQue (Courant != Vide) Faire {  
        Courant = L⇒Places[Courant]⇒Suivant;  
        Compteur = Compteur + 1;  
    }  
    FinTantQue;  
  
    Retourne Compteur;  
}
```

20/09/2006

36

Listes > Algorithme avec allocation contiguë > Supprimer

Premier → 0
Longueur → 6

Position	Elt	Libre	Suivant
0	Dupond	Faux	3
1	Paul	Faux	Vide
2	Toto	Faux -Vrai	3 Indéfini
3	Titi	Faux	1
4	Indéfini	Vrai	Indéfini
5	Indéfini	Vrai	Indéfini

20/09/2006

37

Listes > Algorithme avec allocation contiguë > Supprimer

■ Supprimer : Liste ⊗ Entier ⇒ Liste

Fonction supprimer(L : Liste; Pos : Entier) :
Liste; {...?}

20/09/2006

38

Listes > Algorithme avec allocation contiguë > Insérer

Premier → 0 Longueur → 6

Position	Elt	Libre	Suivant
0	Dupond	Faux	2
1	Paul	Faux	Vide
2	Toto	Faux	3 4
3	Titi	Faux	1
4	Nouveau	Vrai	Indéfini
		Faux	3
5	Indéfini	Vrai	Indéfini

20/09/2006

39

Listes > Algorithme avec allocation contiguë > Insérer

■ Insérer : Liste ⊗ Entier ⊗ Element ⇒ Liste

Fonction Insérer(L : Liste; Pos : Entier; EltIns :
Element) : Liste; {
Compteur : Entier;
Courant : Entier;
PlaceIns : Entier
Compteur = 1;
Ins : Place;
Compteur2 = 0;
PlaceIns = vide;

20/09/2006

40

Listes > Algorithme avec allocation contiguë > Insérer

```
TantQue (Compteur2 < Longueur) Et
(PlaceIns = Vide) Faire {
  Si (L⇒Places[Compteur2]⇒Libre) Alors
  {
    PlaceIns = Compteur2;
  }
  FinSi
  Compteur2 = Compteur2 +1;
}
FinTantQue
```

20/09/2006

41

Listes > Algorithme avec allocation contiguë > Insérer

```
Si (PlaceIns != vide){
  Si (Pos == 0) Alors {
    Ins⇒Element = EltIns;
    Ins⇒Suivant = L⇒Premier;
    Ins⇒Libre = Faux;
    L⇒Premier = PlaceIns;
  }
  Sinon{
    Courant = L⇒Premier;
    TantQue (Courant != Vide) Et (Compteur < Pos) Faire {
      Compteur = compteur + 1 ;
      Courant = Valeur[Courant]⇒Suivant;
    }
    FinTantQue;
  }
}
```

20/09/2006

42

Listes > Algorithme avec allocation contiguë > Insérer

```
Si (Courant != Vide) Et (Compteur == Pos) Alors {
  L⇒Places[PlaceIns]⇒Element = EltIns;
  L⇒Places[PlaceIns]⇒Suivant =
  L⇒Places[Courant]⇒Suivant;
  L⇒Places[PlaceIns]⇒Libre = Faux;
  L⇒Places[Courant]⇒Suivant = PlaceIns;
}
FinSi
}
FinSi
}
FinSi
Retourne L
}
```

20/09/2006

43

Listes > Algorithme avec allocation contiguë > Insérer > Trace

- Insérer(Liste_2, 2, {192.168.1.50,...}) ?
- Insérer(Liste_2, 5, {192.168.1.50,...}) ?

20/09/2006

44

Listes > Algorithme avec allocation contiguë > Succ

Premier → 0
Longueur → 6

Position	Elt	Libre	Suivant
0	Dupond	Faux	2
1	Paul	Faux	Vide
2	Toto	Faux	3
3	Titi	Faux	1
4	Indéfini	Vrai	Indéfini
5	Indéfini	Vrai	Indéfini

20/09/2006

45

Listes > Algorithme avec allocation contiguë > Succ

■ Succ : Place ⇒ Place

```
Fonction Succ(L : Liste; P : Place) :  
Place];{  
  Resultat : Place;  
  Si (P⇒Suivant != Vide) Alors  
    Resultat = L⇒Places[P⇒Suivant];  
  FinSi  
  Retourne Resultat;  
}
```

20/09/2006

46

Listes > A retenir

- ❑ Plusieurs représentations internes possibles.
- ❑ Attention aux cas d'erreur : liste vide, accès en dehors des bornes...

20/09/2006

47