TAD File

Structure **FIFO**

(**F**irst **I**n **F**irst **O**ut)

Description

- Correspond à la notion usuelle de **file d'attente** : file d'attente à un guichet, au RU, au cinéma… En anglais « **queue** »…
- Liste dans laquelle les ajouts se font à une extrémité (fin de file) et les suppressions à l'autre (tête de file).
- **Structure FIFO** : le premier élément entré (**F**irst **I**n) est le premier sorti (**F**irst **O**ut).

Description « schématique » d'une File

Primitives

type **TFile** = **File** de TInfo

bool **fileVide** () const

TInfo **valeurPremier** ( ) const

void **enfiler** (TInfo elem)

void **defiler** ()

Principales utilisations

Les systèmes d'exploitation ont recours à la structure de données « file d'attente » pour gérer l'accès à une ressource partageable :

- file des requêtes d'impression (serveur d'impression)
- file des processus en attente d'un processeur (scheduler ou ordonnanceur)
- …

```
void inverserFile(File<int> & f) {
   Pile<int> p;
   while(!f.fileVide()){ // tant que f n'est pas vide
      p.empiler(f.valeurPremier());
      f.defiler();
   }

   while(!p.pileVide()){ // tant que p n'est pas vide
        f.enfiler(p.valeurSommet());
        p.depiler();
   }
}
```

TAD Liste

Description « schématique »

une **liste** **1er** **2ème** **dernier**

Liste (flèche) **Liste** (flèche vers élément suivant)

Tête (élément)

pas de **suivant**

Description plus « algorithmique »

- En pratique, on va **distinguer le 1er élément** : les insertions/suppressions se feront en **tête de liste**, ou **après** tel ou tel élément.

  ––> notion de **tête de liste** : désigne le 1er élément.

- Tout élément, sauf le dernier, a un **suivant**.

  ––> notion de **flèche** : « pointe » sur l'élément suivant.

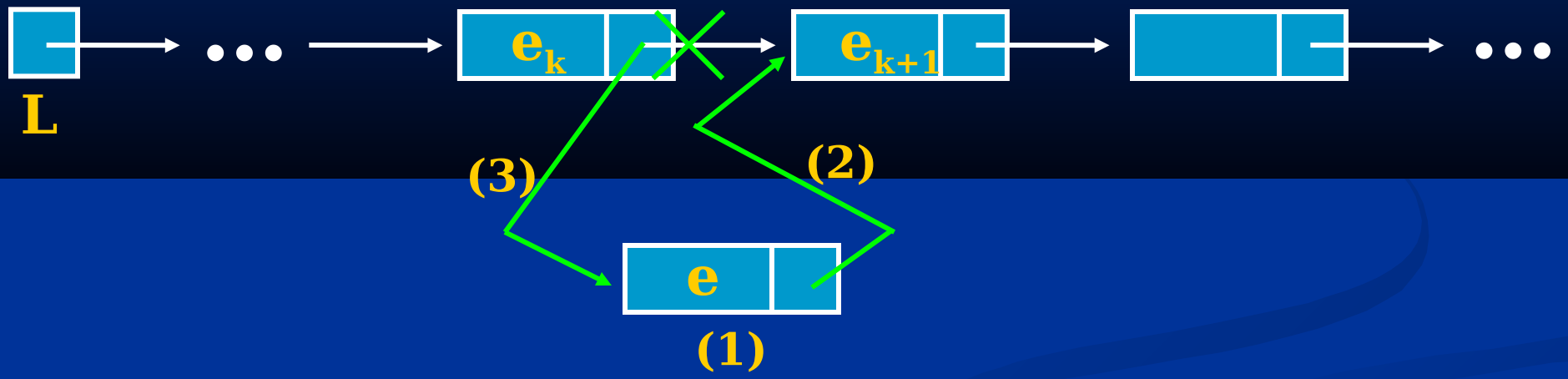
  Rem : il n'y a pas (pour le moment) de notion de précédent (cf. listes doublement chaînées)

Remarques

- La **position physique** des éléments d'une liste est **indépendante** de **l'ordre logique** au sein de la liste : cet **ordre** est uniquement lié au mécanisme des flèches.
- Ainsi, pour **modifier l'ordre** des éléments, il suffit de **modifier les flèches** !

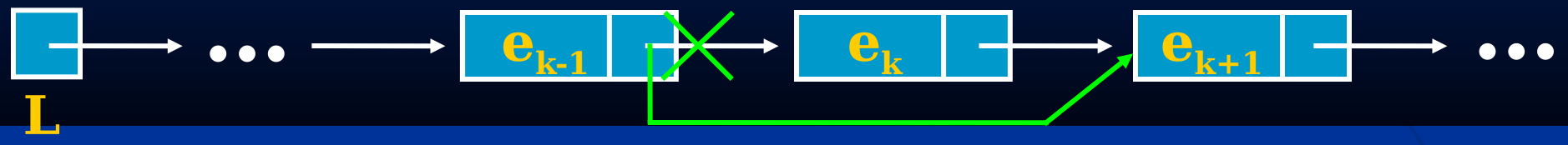
  Exemple : insertion, suppression d'un élément…

Insertion



- **insertion de l'élément e dans la liste L :**
  - **soit insertion en tête de la liste L**
  - **soit insertion après l'élément e_k**
- **<u>remarque</u> : les primitives d'insertion réaliseront les opérations dans l'ordre indiqué (1), puis (2) et finalement (3)…**

Suppression



- **suppression de** l'élément e_k appartenant à la liste **L** :
  - soit **suppression en tête** de la liste L
  - soit **suppression après** l'élément e_{k-1}

Le type TAdresse

Le type **TAdresse** matérialise la notion de flèche :

- intuitivement, la **valeur** d'un objet de type **TAdresse** correspond à **l'adresse physique** d'un objet de la liste

- constante **NULL** de type **TAdresse** : valeur particulière pour une **flèche** ne « pointant » sur rien (pas de suivant) : flèche associée au dernier élément d'une liste

Primitives

type TListe = Liste de TInfo

TAdresse adressePremier() const
TAdresse adresseSuivant() const

TInfo valeurElement(Tadresse adr) const
void modifieValeurElement(Tadresse adr,
  Tinfo elem)

void insererEnTete(TInfo elem)
void insererApres(TInfo elem, TAdresse adr)
void supprimerEnTete()
void supprimerApres(TAdresse adr)

Algorithme de parcours

Affichage du contenu d'une liste.

```
void afficheContenu(Liste<int> l ){
   Tadresse adr;
   adr = l.adressePremier()

   while(!adr.null()){ // adr != NULL
        cout << l.valeurElément(adr ) <<endl;
        Adr = l.adresseSuivant(adr );
   }
}
```

Action mystère…

- faire tourner l'action *mystère* sur la liste

(5, 18, 22, 4, 7)

```
void mystere (Liste<int> &l){
   TAdresse adr, suiv;
   adr = l.adressePremier ();
   if(!adr.null()){
        suiv = l.adresseSuivant ( adr )
        while(!suiv.null()){
            l.insererEnTete(l.valeurElement(suiv));
            l.supprimerApres(adr);
            suiv = l.adresseSuivant ( adr );
         }
   }
}
```

mystère ≡ inverserListe

- ***Principe de l'algorithme :*** insérer en tête de la liste l'élément suivant le premier élément de l (s'il existe), le supprimer, puis passer au suivant.

- ***Remarque :*** il serait possible d'utiliser une **file**, mais…

trop coûteux (en temps et espace mémoire)

…

Recherche dans une liste : principes généraux

Recherche globale

<Initialisations>

Tant que <pas trouvé> et <pas à la fin>
  faire

  Si <élément cherché>==<élément courant>

    Alors <on a trouvé>

  Sinon <on passe au suivant>

Algorithme « recherche dans une liste »

```
TAdresse recherche (Liste<int> l, TInfo elem){
   // retourne l'adresse de elem dans la liste l,
   // NULL si elem n'est pas présent.

   TAdresse adr = l.adressePremier();
   bool trouve = false;

   while(!trouve && !adr.null()) { //adr != NULL
      if(l.valeurElement(adr) == elem)
         Trouve = true;
      else adr = l.adresseSuivant(adr) ;
   }
   return adr;
}
```

Algorithme « insertion d'un élément en fin de liste »

```
void insereElementFin (Liste<int> & l, Tinfo elem)
{
   TAdresse adr, adrPrec;
   adr = l.adressePremier();
   while( !adr.null()) {
      AdrPrec = adr;
      Adr = l.adresseSuivante(adr);
   }
   if(adrPrec.null())  // l ne contient pas d'élément
      l.insererEntete(elem);
   else
      l.insererApres(elem, adrPrec);
}
```

Algorithme « Concaténation de 2 listes dans une 3ème »

```
void
concatener(Liste<int> l1, Liste<int> l2, Liste<int>
    &l3){
   TInfo fictif;
   TAdresse adr, suiv;

   // préparation de la liste L3
   l3.insererEnTete(fictif) // éviter les tests pour
   « insérerEnTête »
   suiv = l3.adressePremier();

   // recopie de la liste L1
   adr = l1.adressePremier();
   while( ! adr.null()) {
      l3.insererApres(l1.valeurElement(adr), suiv);
      suiv = l3.adresseSuivant(suiv);
      adr = l1.adresseSuivant(adr) ;
   }
```

```
    // recopie de la liste L2
    adr = l2.adressePremier() ;
    while( !adr.null()){
      l3.insererApres( l2.valeurElement(Adr), suiv );
      suiv = l3.adresseSuivant(suiv);
      adr = l2.adresseSuivant (adr);
     }

    // correction la liste L3
    l3.supprimerEnTête(L3)     // suppression de l'élément
    fictif
}
```