# Chapitre 1

# Piles, Files et Listes

## 1.1 Introduction

Les ensembles forment la structure de données la plus simple, nous allons rajouter une information en donnant un ordre sur les éléments. D'où la notion de piles et de files.

## 1.2 Piles

### 1.2.1 Les opérations élémentaires

- sommet($p$)
- depile($p$)
- empile($x, p$)
- pilevide()
- vide($p$)

## 1.3 Files

### 1.3.1 Les opérations élémentaires

- tete($f$)

- corps($f$)
- enfile($t, x$) (ou adjq($t, x$))
- filevide()
- vide($f$)

Rien à dire de plus que pour les ensembles, c'est même plus simple. Notons qu'on peut programmer les files à l'aide des piles mais que la réciproque n'est pas vraie.

### 1.3.2 Le jeu de bataille

#### Principe

Il s'agit d'écrire un programme qui simule le jeu de cartes de bataille.
Il doit **distribuer** les cartes, puis **afficher** la séquence correspondant à une partie et finalement **déterminer** quel est le gagnant.

#### Déroulement d'une partie

Chaque joueur a une file de cartes, ils posent chacun la tête de leur file sur la table. Celui qui a la plus grande carte ramasse les cartes, d'abord celle de son adversaire puis la sienne et les met en queue de sa file. Si les deux cartes ont les mêmes valeurs, chacun des joueurs place à l'envers la tête de sa file sur sa pile sur la table puis la tête de sa file sur sa pile. Celui qui a la plus grande carte ramasse comme précédemment les deux piles. Est déclaré perdant celui qui n'a plus de carte dans sa file. Lorsque les deux n'ont plus de carte il y a égalité.

#### Programmons

**si** lui **et** moi **alors** egalite
**si** mafile **est vide alors** lui
**si** safile **est vide alors** moi

jegagne = macarte > sacarte
ilgagne = macarte < sacarte

mafile = **si** jegagne **alors**
mafile = enfile(mafile,sapile)
mafile = enfile(mafile,mapile)
safile = **si** ilgagne **alors**
safile = enfile(safile,mapile)
safile = enfile(safile,sapile)
mapile = empile(macarte,mapile)
sapile = empile(sacarte,sapile)

macarte = tete(mafile)
sacarte = tete(safile)

mafile = corps(mafile)
safile = corps(safile)

etc..

## 1.4 Listes

Les listes sont construites à partir des files en ne permettant pas l'adjonction en queue mais des opération spécifiques. Citons :

- les listes **triées** et qui le restent après insertion (insav($x, l$) ou insap($l, x$)).
- les listes **circulaires** que nous appelerons des carrousels (voir plus loin).
- les listes **doublement chaînées**. Notons ici que ce type de liste n'apporte rien de neuf si ce n'est une exécution plus rapide du fait de l'opérateur *précedent()*.
- etc., on peut imaginer tout ce qu'on veut pour améliorer l'exécution de l'une ou l'autre des opérations (adjonction, recherche, suppression, etc.).

## 1.5 carrousel

### 1.5.1 Les opérations élémentaires

**sorte** carroussel;

**utilise** T;

**pour tout** x **de type** T **et tout** carrousel c
**on dfinit**:

carrousel carrouselvide()
carrousel rotation($c$)
carrousel inclusion($x, c$)
carrousel extraction($c$)
T element(c)

### 1.5.2 Flavius

Avec cela il est facile de programmer Flavius ...

*"... les méchants romains voulaient tuer 39 des 40 esclaves qu'ils avaient capturés. Ils les font mettre en cercle, comptent 1, 2, 3, 4, 5, 6 et tuent le septieme, puis comptent etc.. Où le futé Flavius, esclave parmi les 40, devait-il se mettre?[1] "*

**int** Flavius()

{
carrousel $c$;

$c$=carrouselvide();

[1]voir sujet d'examen J.F.Dufourd ou DNA janvier 1989

**for** $(i = 1; i <= 40; i++)$ $c$=inclusion$(i, c)$;

**for** $(i = 1; i <= 39, i++)$
{
**for** $(k = 1; k <= 7; k++)$ rotation$(c)$;
extraction$(c)$;
}
**imprimer**("la bonne place est :",element(c));
}

Voici à titre éducatif le programme C récursif qui tourne sans structure de données particuliaire:

main()

{
**printf**("le bon numero est :%d\ n", bonnumero(0,40)+1);
}

**int** bonnumero$(i, n)$ /* on a $n$ elements ,
on vient d'eliminer
celui qui est situe
avant le numero $i$ */

**int** $i, n$ ;

{
**int** $k$,$v$;

**if** $(n == 1)$ **return**(0);

$v$=$((i-1)+7)\%n$;
$k$=bonnumero(v, $n-1$);

**if** $(k < v)$ **return**$(k)$;
**if** $(k >= v)$ **return**$(k+1)$;
}

### 1.5.3 Serveur d'imprimantes

Un serveur d'imprimante est un programme qui tourne sur un ordinateur sur lequel sont connectés:

- un réseau par où arrivent les informations à imprimer on supposera que l'on reçoit des paquets de la forme:

  **o i t c**

  où

  - **o** est un nom d'ordinateur
  - **i** est un nom d'imprimante
  - **t** est *vrai* ou *faux* suivant que le caractère **c** est une commande ou du texte.
  - **c** est le caractère ou la commande

- un ensemble d'imprimantes sur des ports RS232 normaux. (l'imprimante **i** sera connectée au port **i**)

Sur le réseau peuvent être connectés autant d'ordinateurs qu'on veut dans la mesure où ils ont tous un nom différent. On suppose aussi pour simplifier que tout se passe sans erreurs.
D'après le protocole suivant:

**repeter indefiniment**

- **lecture** eventuelle d'un paquet en reception
  (si yaPAQUET (variable systeme) est vrai)

- **emission** d'un caractere sur une imprimante i
  (si impLIBRE(i) est vrai
  (variable systeme associee au port i))

**fin repeter**

Notons bien qu'on ne peut pas émettre plus d'un caractère sur une seule imprimante entre deux lectures sur le réseau.
L'impression d'un message par un ordinateur consiste à envoyer:

- la commande **sot** (**S**tart **O**f **T**exte)

- les caractères composant le texte un par un

- la commande **eot** (**E**nd **O**f **T**exte)

Un même ordinateur peut envoyer simutanément des fichiers vers des imprimantes différentes. Plusieurs ordinateurs peuvent envoyer simultanément des fichiers vers la même imprimante.
Stuctures de données nécessaires:

- Il nous faut un **ensemble de triplets** (**o**rdinateur-**i**mprimante-**f**ichier), le fichier étant une liste de caractères. Ce triplet (**o, i, f**) n'existe que pendant **sot** ... **eot**.

- Il nous faut une **liste** par imprimante **triée** en fonction de priorités de tailles pour les fichiers en attente d'impression.

- Il nous faut un **carrousel** de **fichiers** en sortie pour émettre vers les imprimantes.

On peut maintenant écrire:

```
Reception_paquet(paquet p)

switch commande(p):
case SOT: ens3=inclu(triplet(o,i,fichvide()), ens3)
case EOT: lfai(i)=insere(fichier(o,i,ens3),lfai(i))
  ens3=retire(fichier(o,i,ens3),ens3)
case CAR: fichier(o,i,ens3)=adjq(fichier(o,i,ens3),c)



Emission_caractere()

cs=rotate(cs) ! carrousel en sortie
i=imprimante(cs)
si non impLIBRE(i) return

si vide(fs(i)) alors
si non vide(lfai(i))
fs(i)=tete(lfai(i)); lfai(i)=corps(lfai(i));

si non vide(fs(i)) alors
imprime(tete(fs(i)); fs(i)=corps(fs(i));
```