Cours
La structure de données tableau et quelques algorithmes

- La structure de donnée tableaux
- Evaluation de polynômes, recherche de minimum
- Les algorithmes de tri
- Les algorithmes de recherche séquentiel et dichotomique

Tableaux: déclaration/initilisation

Tableau= collection de données homogènes, accessibles par un indice entier.

Déclaration d'un *type* tableau

```
type <nom> = tableau de <valeur> <nom de type>
```

Exemple :

```
constante N = 5
type polynome = tableau de N réels
```

On peut alors déclarer une variable de type tableau
Exemple:

```
p : polynome
```

et même l'initialiser (p représente $3x^4 + 2x^3 + 1$)

```
p : polynome = {1.0, 0.0, 2.0, 3.0, 0.0}
```

Tableaux: déclaration/initilisation

Ensuite, on peut accéder au i-ème élément du tableau en écrivant

```
p[i]
```

ATTENTION: selon les langages de programmation, les tableaux commencent:

- à l'indice 1 (pseudolangages, Pascal)
- à l'indice 0 (en C et C++)

Avantage: pouvoir accéder aux éléments avec une adresse calculée.
Contrainte: tous les éléments doivent être du même type.[1]

---

[1] On verra plus avant les "structures", qui ont les avantages et desavantages inverses.

Lire et écrire des tableaux

Pour lire ou écrire un tableau a de N éléments, on utilisera des boucle:

lecture

```
pour i<-1 à N faire
     ecrire "entrez l'élément " i
     lire p[i]
fin pour
```

écriture

```
pour i<-1 à N faire
     ecrire p[i]
fin pour
```

Dans la suite, on écrira juste "*lire* le tableau p" et "*ecrire* le tableau p"

Exemple: Calcul de la valeur d'un polynome en un point

On veut évaluer le polynôme

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

au point $x = k$.

Première méthode

On évalue la somme en calculant pour $i = 1, \ldots, N$:

1. La puissance k^i
2. Le produit $a_i k^i$ (c'est à dire $p[i+1]k^i$ dans le programme)

Pour que le calcul de k^i soit rapide, on aura avantage à calculer **de droite à gauche**:

$$1, k, k^2, \ldots, k^n$$

Le calcul du polynome se fait **de droite à gauche**.

```
p : polynome = {1.0, 0.0, 2.0, 3.0, 0.0}
i : un entier
v,a,p : trois réels
ecrire entrez le point d'évaluation
lire a
v <- p[1]
p <- a
pour i <- 2 a N faire /* Inv: v=sum(k=1,k=i-1) p[k]*a^(k-1), p=a^(i-1) */
    v  <- v + p[i]*px
    p <- p*a
fin faire
ecrire "la valeur est " v
```

N.B.: On fait $2N$ multiplications et N additions. On peut faire mieux!

Methode de Horner (N multiplications, N additions)

On utilise la factorisation:

$$p(x) = (\ldots(a_n x + a_{n-1})x + \ldots + a_1)x + a_0$$

Le calcul se fait **de gauche à droite**.

```
p : polynome = {1.0, 0.0, 2.0, 3.0, 0.0}
i : un entier
v,x : deux réels
ecrire entrez le point d'évaluation
lire a
v <- p[N]
pour i <- 1 a N-1 faire /* Inv: v=sum(k=1,k=i-1) p[k]*a^(N-i) */
    v<-v*a + p[N-i]
fin faire
ecrire "la valeur est " v
```

Recherche du minimum

```
/* calcule l'indice imin du minimum du segment du tableau a */
/* a partir de i  (i.e. a[imin]=min(a[i]...a[N]))     */
k,i,imin : trois entiers

imin <- i
pour k <- i+1 a  N faire
   si (a[k] < a[imin])
   alors imin <- k
   fin si
fin pour
```

Tri par séléction

Donnée: un tableau a de N nombres entiers $a[1], \ldots, a[N]$.

Résultat: Réarrangement croissant $b[1], \ldots, b[N]$ des nombres $a[i]$.
Exemple:

$$a = \begin{array}{|c|c|c|c|c|}\hline 24 & 5 & 81 & 2 & 45 \\ \hline\end{array}$$
$$b = \begin{array}{|c|c|c|c|c|}\hline 2 & 5 & 24 & 45 & 81 \\ \hline\end{array}$$

Principe: A la i-ème étape, on cherche le minimum de $a[i], a[i+1], \ldots, a[N]$ et on l'échange avec $a[i]$.
Il faut *toujours* $n(n+1)/2$ itérations.

Tri par sélection: le programme

```
i, imin, tmp, k: quatre entiers
lire le tableau a

pour i <- 1 a N-1 faire
 imin <- i /* cherche index du minimum de a[i]...a[n] */
 pour k <- i+1 a  N faire
  si (a[k] < a[imin]) alors imin <- k fin si
 fin pour
 tmp <- a[i]       /* echange a[i] et a[imin] */
 a[i] <- a[imin]
 a[imin] <- tmp
fin pour
ecrire le tableau a
```

Recherche d'un élément (méthode naïve)

```
/* calcule le premier j tel que a[j]= e, s'il existe */
   i : un entier
   trouve : un booléen

   trouve <- faux
   i <- 0
   tant que  ((trouve = faux) et i<N) faire
      i <- i+1
      si  (a[i] = e)
      alors trouve <- vrai
      fin si
   fin tant que
   si (trouve=vrai) alors  ecrire i fin si
```

Invariant et complexité

- proposez un invariant pour ce programme (c'est un peu complexe)

Cas le pire $a[i] = e$ toujours faux. On fait alors N iterations.

Cas moyen on a $\frac{1}{N}\sum_{i=1}^{i=N} i = \frac{N+1}{2}$ itérations

On peut faire mieux.

Recherche d'un élément (méthode dichotomique)

dichotomique: de $\delta\upsilon o$ (deux) et $\tau o\mu\eta$ (coupe).
La méthode que vous utilisez tous pour chercher un numéro de téléphone dans l'annuaire.

Avantage: on peut *toujours* faire en $log_2(n)$
Contrainte: le tableau doit etre trié

MÉTHODE DICHOTOMIQUE: LE PROGRAMME

```
/* calcule j tel que a[j]= e, s'il existe */
   bas,haut,milieu : trois entiers
   trouve : un booléen

   trouve <- faux
   bas <- 1
   haut <- N
   /* Inv: si présent, e se trouve entre bas et haut   */
   /* Terminaison: haut - bas diminue à chaque itération */
   tant que  ((trouve = faux) et haut>=bas) faire
      milieu < (bas+haut)/2
      si  (a[milieu] = e)
      alors trouve <- vrai
```

```
    sinon
      si (a[milieu] < e)
      alors bas<-milieu+1
      sinon haut<-milieu-1
      fin si
    fin si
  fin tant que
  si (trouve=vrai) alors ecrire milieu fin si
```

Exercice: recherche du zéro d'une fonction On cherche le zéro d'une fonction continue f dans l'intervalle $[a, b]$, en sachant que $f(a) * f(b) < 0$. Utilisez la méthode dichotomique, avec un paramètre ϵ donnant la valeur en dessous de laquelle on considère que l'on a atteint le zéro.
N'oubliez pas de proposer un invariant.