Programmation en C et Méthodes Numériques

Tableaux et Matrices

T. Dietenbeck (thomas.dietenbeck@upmc.fr)

Université Pierre et Marie Curie

UPMC
SORBONNE UNIVERSITÉS

UPMC SORBONNE UNIVERSITÉS

Plan

UPMC SORBONNE UNIVERSITÉS

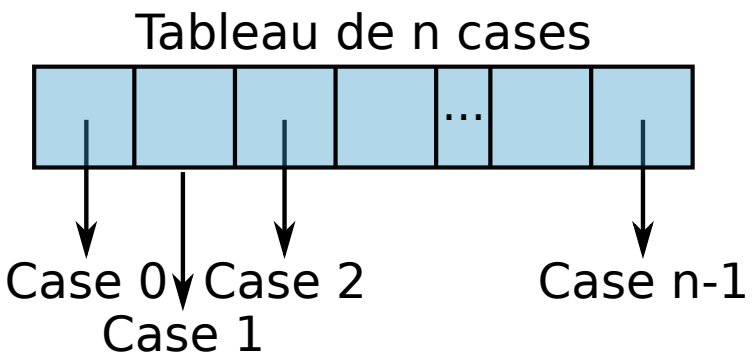
Plan

Qu'est ce qu'un tableau

Définition

Ensemble

- de taille fixe
- de variables du même type
- adressées par un indice (ou numéro) : leur position dans le tableau
- contigüe en mémoire



Qu'est ce qu'un tableau

Définition

Ensemble

- de taille fixe
- de variables du même type
- adressées par un indice (ou numéro) : leur position dans le tableau
- contigüe en mémoire

Exemple

- Résultats de plusieurs lancers de "Pile ou Face" : tableau de booléens
- Numéros d'étudiants d'un groupe de TD : tableau d'entiers
- Coordonnées d'un vecteur : tableau de réels
- Plaque d'immatriculation des véhicules d'une entreprise : tableau de chaines de caractères

UPMC SORBONNE UNIVERSITÉS

Utilisation d'un tableau

2 étapes

1. Déclaration
   - Nom de la variable
   - Type de données stockées
   - Taille du tableau
2. Utilisation
   - Remplissage
   - Parcours
   - Affichage
   - ...

Déclaration d'un tableau

Déclaration

- Nom de la variable
- Type de données stockées
- Taille du tableau

Syntaxe

```
typeDonnee nomTab[tailleTab];
```

Exemple

```
// Un tableau de 42 entiers
int tabEntier[42];  // Son contenu est inconnu pour le moment

// Un tableau de 4 entiers
int tabEntier2[] = {1, 3, 3, 7};  // Son contenu est 1, 3, 3, 7
```

UPMC SORBONNE UNIVERSITÉS

Fixer la taille d'un tableau

Variable pour la taille d'un tableau

Il est interdit d'utiliser une variable pour dimensionner un tableau !

```
int taille = 3;
int tab1[taille]= {1, 2, 3};
```

- Comment faire pour limiter le nombre de modifications lorsqu'on change la taille d'un tableau ?

Fixer la taille d'un tableau

Définition de constantes

- En C, on peut associer une valeur à un identificateur

```
#define NOM_CONSTANTE valeur  // Pas de ; a la fin de la ligne
```

- On ne peut pas effectuer d'opérations sur l'identificateur : la valeur correspondante est constante !
- **Remarque :** les constantes se déclarent en dehors des fonctions (soit dans les .h, soit avant le `main`)

Exemple

```
// Definition d'une constante TAILLE qui vaut 42
#define TAILLE 42
...
// Declaration d'un tableau d'entiers de 42 cases
int tab[TAILLE];
```

Utilisation d'un tableau

Accès à une valeur

- On accède au contenu d'une case d'un tableau par `tab[i]`
- Attention : les indices (i) vont de 0 à n-1 (où n est la taille du tableau)

Exemple

```
int tabEntier[ ] = {1, 3, 3, 7};
tabEntier[2] = 42;     // Modification de la 3eme case du tableau
int i = tabEntier[0]; // Copie de la 1ere case dans la variable i

tabEntier[4] = 2;      // Erreur!
```

Remarque

Accéder à la valeur d'une case (dont on connait l'indice) est une opération "élémentaire" qui se fait donc en $\mathcal{O}(1)$

Parcours complet d'un tableau

Parcours de toutes les valeurs d'un tableau

Il faut accèder successivement au contenu de chaque case du tableau

$\Rightarrow$ Utilisation d'une boucle `for`

```
// Boucle de la 1ere case (0) a la derniere (TAILLE-1)
for( i = 0; i < TAILLE; i++ ) {
  /* Serie d'instructions */
}
```

Exemples

- Affichage des valeurs
- Copie / égalité des valeurs du tableau
- Recherche de la valeur minimum / maximum
- Calcul de la somme / moyenne des valeurs

Remarque

Parcourir un tableau est de complexité $\mathcal{O}(n)$ (n accès à une case)

Parcours complet d'un tableau

Parcours de toutes les valeurs d'un tableau

Il faut accèder successivement au contenu de chaque case du tableau

⇒ Utilisation d'une boucle `for`

```
// Boucle de la 1ere case (0) a la derniere (TAILLE-1)
for( i = 0; i < TAILLE; i++ ) {
  /* Serie d'instructions */
}
```

Exemples

```
for( i = 0; i < TAILLE; i++ ) {
  // Affichage de toutes les valeurs
  printf( "%d  ", tab[i] );
}

int maxT = 0;
for( i = 0; i < TAILLE; i++ ) {
  if( maxT < tab[i] )  // Recherche du max
    maxT = tab[i];
}
```

Parcours partiel d'un tableau

Parcours partiel des valeurs d'un tableau

Il faut accèder successivement au contenu de chaque case du tableau sans connaitre le nombre de répétition

⇒ Utilisation d'une boucle `while`

```
int i = 0;
while( (i < TAILLE) && (condition d arret) ) {
  /* Serie d instructions */
  i++;
}
```

Exemples

- Recherche d'une valeur dans le tableau
- Parcours d'un tableau surdimensionné

Parcours partiel d'un tableau

Parcours partiel des valeurs d'un tableau

Il faut accèder successivement au contenu de chaque case du tableau sans connaitre le nombre de répétition

⇒ Utilisation d'une boucle `while`

```
int i = 0;
while( (i < TAILLE) && (condition d arret) ) {
  /* Serie d instructions */
  i++;
}
```

Exemple : Recherche du premier 5 dans un tableau d'entier

```
int i = 0;
while( (i < TAILLE) && (tab[i] != 5) ) { i++; }
if( i == TAILLE )
  printf( "Pas de 5 dans le tableau\n" );
else
  printf( "Le 1er 5 est a la position %d\n", i );
```

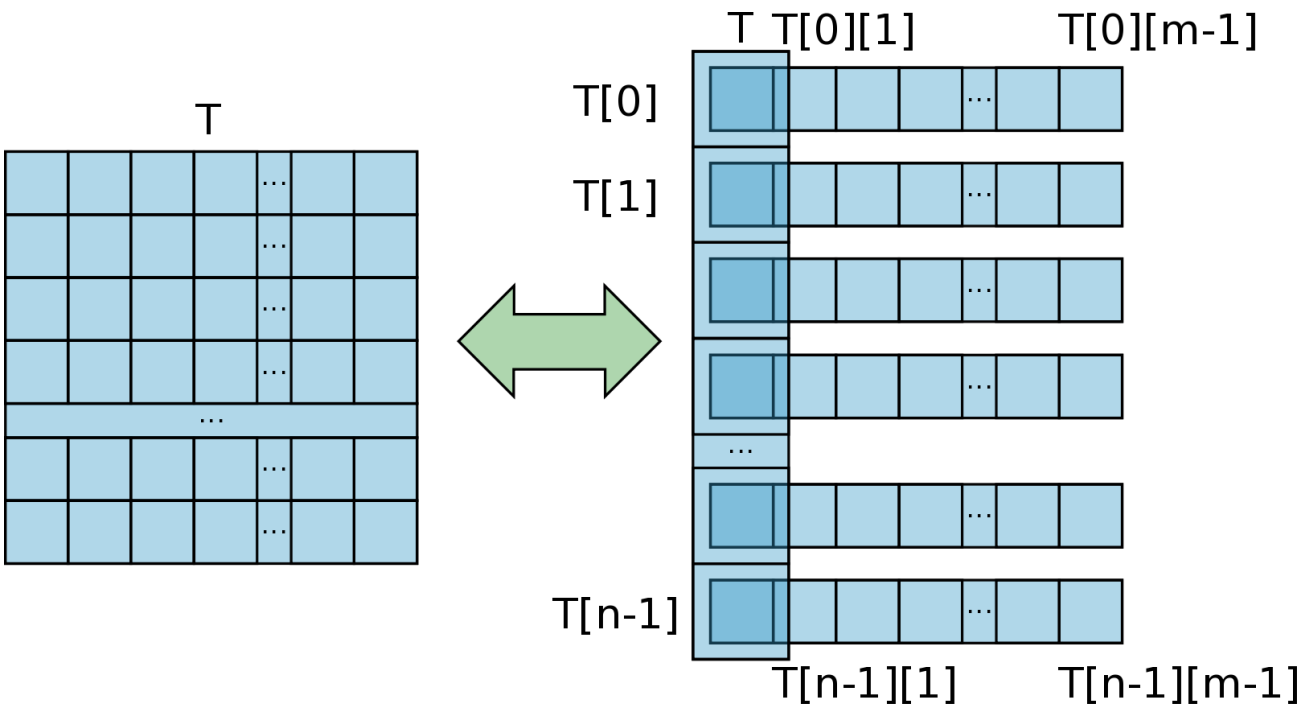
Matrice

Définition

- Un tableau 2D (ou matrice) peut être vu comme un tableau contenant des tableaux contenant des éléments
- Déclaration

```
typeDonnee nomMat[DIMX][DIMY];
```

- Utilisation : accès à la j^{ème} case de la i^{ème} ligne : `mat[i][j]`
- Le parcours de tous les éléments d'une matrice est de complexité $\mathcal{O}(n^2)$



Exemple

Parcours de toutes les valeurs d'une matrice

```
int matEntier[DIM_X][DIM_Y];
  /* Code pour initialiser la matrice */
  ...
  /* */

// Recuperation d'une ligne
int tabEntier[ ] = matEntier[1];

// Affichage de toutes les valeurs
int i, j;
  // Parcours des lignes
for( i = 0; i < DIM_X; i++ ) {
    // Parcours des colonnes
  for( j = 0; j < DIM_Y; j++ ) {
    printf( "%d  ", matEntier[i][j] );
  }
  printf( "\n" );
}
```

UPMC SORBONNE UNIVERSITÉS

Plan

Tableau et Pointeur

Relation entre tableau et pointeur

- Une variable de type tableau contient l'adresse de la première case du tableau

$$\Rightarrow \textbf{c'est un pointeur !}$$

- Pour autant, un pointeur n'est pas un tableau !

Tableau et Fonction

Tableau et fonction

- Si on passe un tableau à une fonction, on passe donc l'adresse de la première case
- C'est un **passage par adresse** (implicite) : on se souviendra de toutes les modifications du tableau survenues dans la fonction

Limitations

- Le résultat d'une fonction ne peut pas être un tableau
- La mémoire est en effet libérée en sortant de la fonction et l'adresse du tableau ne correspondra donc plus à rien
  - Erreur de segmentation : le programme cherche à accéder à une partie de la mémoire qui ne lui est pas (plus) allouée
  - Erreur d'exécution : les données pointées dans le tableau ont été modifiées car la zone mémoire a été utilisée pour une autre portion du programme.

Solution : Allocation dynamique du tableau

Allocation dynamique

Variable et mémoire

- Toute variable nécessite un certain espace en mémoire (*e.g.* 4 octets pour un `int`)
- Cet espace peut être reservé automatiquement par le programme lors de la déclaration de la variable (*i.e.* au début d'une fonction)
- On peut cependant demander explicitement à réserver un espace mémoire (*i.e.* en cours de fonction). C'est l'**allocation dynamique** !

Conséquences

- Un espace mémoire réservé explicitement ne peut pas être libéré automatiquement et permet donc de renvoyer un tableau comme résultat
- Il faut penser à libérer explicitement la mémoire pour éviter les "fuites"

Allocation de mémoire

Mémoire utilisée par un type

- Avant d'allouer de la mémoire, il faut savoir combien en demander !
- L'opérateur `sizeof( type )` renvoie la taille (le nombre d'octets) utilisé par une variable de type `type`

Exemple

```
printf("int: %d octets\n", sizeof(int));
printf("char: %d octets\n", sizeof(char));
printf("double: %d octets\n", sizeof(double));
```

affiche

```
int: 4 octets
char: 1 octets
double: 8 octets
```

UPMC SORBONNE UNIVERSITÉS

Allocation dynamique

Cycle de vie d'un tableau

1. Déclaration
   - Nom de la variable
   - Type de données stockées
2. **Allocation dynamique du tableau**
3. Utilisation
   - Remplissage
   - Parcours
   - Affichage
   - ...
4. **Libération de l'espace mémoire**

Remarques

- À partir de maintenant, les tableaux seront **toujours** obtenus par allocation dynamique
- On ne donne plus la taille du tableau lors de la déclaration mais lors de l'allocation dynamique

Allocation de mémoire

Syntaxe

Il existe 2 fonctions pour réserver de la mémoire :

- `void *malloc( size_t size )` réserve un bloc de `size` octets en mémoire. Renvoie un pointeur vers le premier octet du bloc si l'allocation a réussie et `NULL` sinon. Attention : Le contenu de ce bloc est aléatoire !
- `void *calloc( size_t nmemb, size_t size )` réserve `nmemb` blocs de `size` octets en mémoire. Renvoie un pointeur vers le premier octet du bloc si l'allocation a réussie et `NULL` sinon. Le contenu de chaque bloc est égal à 0.

Remarques

Ces 2 fonctions font partie de la bibliothèque `stdlib.h`, qu'il faut donc penser à inclure au début du programme

Allocation de mémoire

Exemple

```
int i;
// Allocation dynamique d'un tableau de 42 entiers
int* tabEntier = malloc( 42 * sizeof( int ) );
// Important: Initialisation du tableau (que des 0)
for( i = 0; i < 42; i++ ){ tabEntier[i] = 0; }

// Allocation dynamique d'un tableau de 42 reels
float* tabReel = calloc( 42, sizeof( float ) );
// Pas besoin d'initialiser, calloc le fait
```

Libération de la mémoire

Syntaxe

- La fonction `void free( void *ptr )` libère l'espace mémoire pointé par `ptr`.

Remarques

- Il faut que `ptr` ait été obtenu par allocation dynamique !
- Il faut toujours autant de `free` que de `malloc` et de `calloc` dans un programme
- Pour éviter d'utiliser un espace mémoire libéré (ou desalloué), il faut penser à faire pointer `ptr` vers `NULL`.

UPMC SORBONNE UNIVERSITÉS

Libération de la mémoire

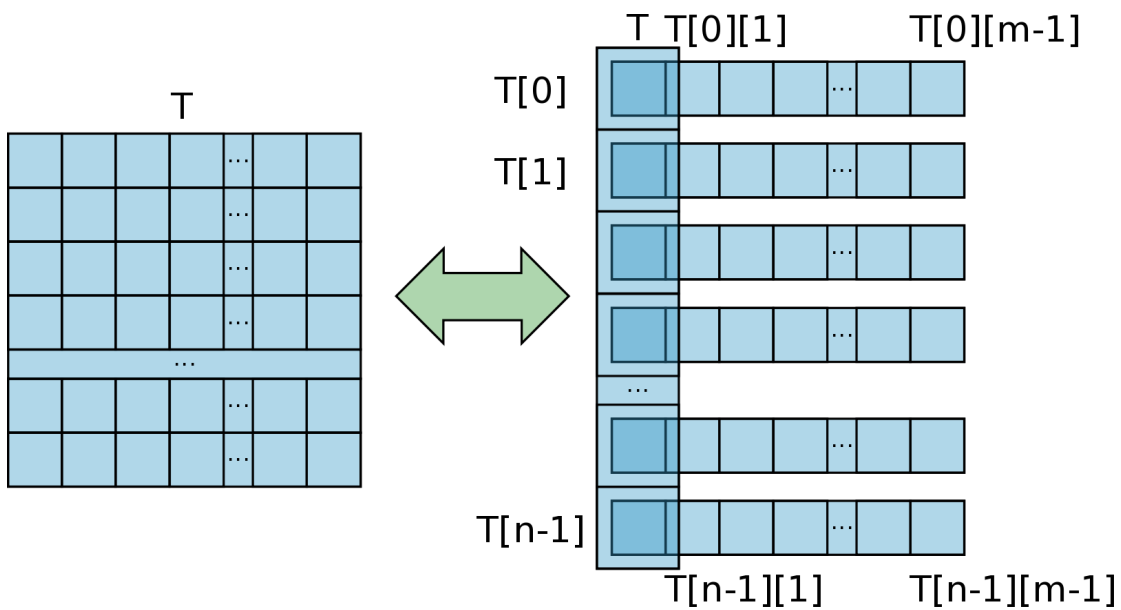
Exemple

```
// Allocation dynamique d'un tableau de 42 entiers
int* tabEntier = malloc( 42 * sizeof( int ) );
// Utilisation du tableau
...

// Liberation de l'espace memoire
free(tabEntier);
// Par securite, on fait pointer tabEntier vers NULL
tabEntier = NULL;
```

Matrices

Principe

- Un tableau 2D (ou matrice) peut être vu comme un tableau contenant des tableaux contenant des éléments
- Il faut donc allouer le tableau "colonne" puis chacun des tableaux "lignes"
- De même, lors de la libération de la mémoire, on libèrera d'abord les tableaux "lignes" puis le tableau "colonne"
- Attention à l'ordre des opérations (libération de mémoire toujours dans le sens inverse de l'allocation) !



Exemple : Allocation

```
#define NB_ROWS 13;
#define NB_COLS 37;
    ...

// 1) Declaration d'une matrice de reels
float** matReel;

// 2a) Allocation dynamique du tableau colonne
  // Attention: bloc memoire de POINTEURS vers des reels
**matReel = malloc( NB_ROWS * sizeof( float* ) );
// 2b) Allocation dynamique de tableau ligne
for( i = 0; i < NB_ROWS; i++ ) {
  // Chaque ligne (matReel[i]) est un tableau de float
  matReel[i] = calloc( NB_COLS, sizeof(float) );
}


// Utilisation de la matrices
matReel[0][0] = 3.14159;
matReel[12][36] = 1.4142;
    ...
```

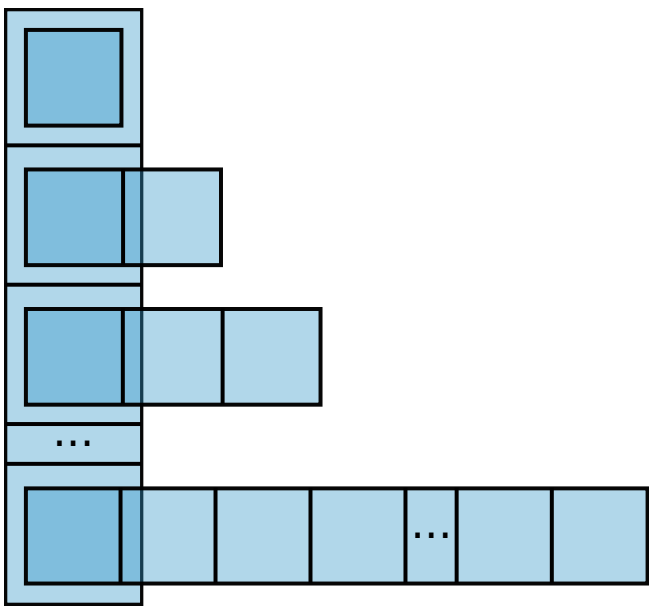
Exemple (suite) : Libération

```
// Utilisation de la matrices
matReel[0][0] = 3.14159;
matReel[12][36] = 1.4142;
    ...

// 1a) Liberation de l'espace memoire des tableaux lignes
for( i = 0; i < NB_ROWS; i++ ) {
  free( matReel[i] );
}
// 1b) Liberation de l'espace memoire du tableau colonne
free( matReel );

// 2) Par securite, on fait pointer matReel vers NULL
matReel = NULL;
```

Exemple plus compliqué

Nombre de colonnes variables

- Chaque ligne d'un tableau 2D n'a pas forcément le même nombre de colonnes
- Exemples :
  - Matrice triangulaire : Triangle de Pascal, Résolution de système d'équation (decomposition LU, de Cholesky, . . .)
  - Notes d'étudiants ayant un contrat partiel
  - Matrice "sparse" (faible nombre d'éléments différents de 0)
- Gain de place en mémoire (on ne stocke que les éléments non nuls)



Exemple plus compliqué

Matrice triangulaire

```
// Declaration
int** matTriangle;
// Allocation dynamique
matTriangle = malloc( DIM_X * sizeof(int*) );  // de la colonne
for( int i = 0; i < DIM_X; i++ )
  matTriangle[i] = calloc( i+1, sizeof(int) );  // Tableau 1D de
      ↪ i+1 elements

// Affichage: attention aux tailles des tableaux
for( int i = 0; i < DIM_X; i++ ) {
  for( int j = 0; j <= i; j++ )
    printf( "%d  ", matTriangle[i][j] );
  printf( "\n" );
}

// Liberation
for( int i = 0; i < DIM_X; i++ )
  free( matTriangle[i] );  // des lignes
free( matTriangle );  // de la colonne
```

Allocation dynamique

Quelques erreurs classiques

```c
#include <stdio.h>
  // Oubli de l'inclusion de stdlib

#define DIM_X 10
#define DIM_Y 15;  // Pas de ; apres un #define

int main( ) {
  int i;

  int *tabEntier;
  tabEntier[0] = 1;  // Utilisation d'un tableau non alloue

  int tabEntier2[DIM_X];
  free( tabEntier2 );  // Destruction d'un tableau non dynamique

  double *tabReel = malloc( DIM_X * sizeof( double ) );
  tabReel[DIM_X+1] = 2;  // Case hors du tableau

  ...
```

Allocation dynamique

Quelques erreurs classiques

```
  ...
  double** matReel = malloc( DIM_X * sizeof( double* ) );
  for( i = 0; i < DIM_X; i++ )
    matReel[i] = calloc( DIM_Y, sizeof( double ) );

  free( matReel );  // Mauvais ordre de destruction
  for( i = 0; i < DIM_X; i++ )
    free( matReel[i] );  // matReel[i] n'est plus accessible

  // Mauvais type dans sizeof
  double** matReel2 = malloc( DIM_X * sizeof( double ) );

  double *tabReel2 = malloc( DIM_X * sizeof( double ) );
  // Oubli de la liberation de tabReel2 (et tabReel)

  return 0;
}
```

UPMC SORBONNE UNIVERSITÉS

Plan

UPMC SORBONNE UNIVERSITÉS

Rappels

Tableau et fonction

- Si on passe un tableau à une fonction, on passe l'adresse de la première case
- C'est un **passage par adresse** (implicite) : on se souviendra de toutes les modifications du tableau survenues dans la fonction
- Le résultat d'une fonction ne peut pas être un tableau **créé automatiquement**

UPMC SORBONNE UNIVERSITÉS

Rappel : Passage de paramètres

Syntaxe

- Pour un tableau **alloué dynamiquement**, seule la notation suivante est autorisée

```
typeRetour nomFonction( typeDonnee* tab, ... )
```

Exemple

```
/** Echange de 2 cases d'un tableau */
void echangerPos( int* tab, int dim, int a, int b ) {
  int c;
  // On s'assure que les positions echangees sont dans le tableau
  if( (a >= 0) && (a < dim) (b >= 0) && (b < dim) ){
    // On echange les 2 positions
    c = tab[a];  // en passant par une variable intermediaire
    tab[a] = tab[b];    tab[b] = c;
  }
}

/** Liberation de l'espace memoire et pointage vers NULL */
void freeTab( int* tab ) {
  free( tab );    tab = NULL;
}
```

Tableau comme résultat de fonction

Syntaxe

- L'allocation dynamique permet de renvoyer un tableau comme résultat d'une fonction
- Le tableau étant alloué dynamiquement, seule la syntaxe * est acceptable

```
typeRetour* nomFonction( ... )
```

Exemple

```
/** Allocation et initialisation d'un tableau */
int* init( int dim ) {
  int i, *tab;

  // On cree un tableau d'entiers de taille dim
  tab = malloc( dim * sizeof( int ) );
  // On met des 0 dans toutes les cases
  for( i = 0; i < dim; i++ ) { tab[i] = 0; }

  return tab;  // On renvoie le resultat
}
```

UPMC SORBONNE UNIVERSITÉS

Plan

UPMC SORBONNE UNIVERSITÉS

Exercice : Enoncé

Racine d'un polynôme

- Soit $p : x \mapsto \sum_{i=0}^{n} a_i x^i$ un polynôme de degré n. On suppose que ses coefficients a_i sont stockés dans un tableau p par ordre croissant de puissance (*i.e.* a_0 est dans $p[0]$ et a_n dans $p[n]$).
- On cherche à calculer une de ses racines par la méthode de Newton : $x_{n+1} = x_n - p(x_n)/p'(x_n)$ sachant x_0

Question

- Écrire un programme permettant de calculer x à une précision ε
- Quelle est sa complexité ?

UPMC SORBONNE UNIVERSITÉS

Exercice : Analyse

Entrées et Sorties

- Données :
  - degré n et coefficients a_i du polynôme
  - valeur de départ x_0
  - précision souhaitée ε
- Résultat : racine du polynôme x

Fonctions

- `valPolynome` : calcul de $p(x)$
- `valDerivee` : calcul de $p'(x)$

UPMC SORBONNE UNIVERSITÉS

Exercice

1 au début (CodinGame)

- À partir d'une suite de 1 et de 0, on souhaite réunir tous les 1 au début de la liste en un minimum d'opérations.
- Une opération se définit par l'échange de deux éléments situés à des positions différentes.

Exemple : Passer de 0011101010111000000 à 1111111000000000 nécessite 4 opérations

Question

- Écrire un programme qui calcule le nombre minimum d'échanges permettant d'obtenir la liste correctement ordonnée.

UPMC SORBONNE UNIVERSITÉS

Exercice

1 au début (CodinGame)

- À partir d'une suite de 1 et de 0, on souhaite réunir tous les 1 au début de la liste en un minimum d'opérations.
- Une opération se définit par l'échange de deux éléments situés à des positions différentes.

Exemple : Passer de 0011101010111000000 à 1111111000000000 nécessite 4 opérations

2 solutions

1. Parcours de la chaîne dans les 2 sens
   - de gauche à droite pour trouver les 0
   - de droite à gauche pour trouver les 1

   jusqu'à ce que les 2 positions se croisent
2. Comptage du nombre n de 1 dans la séquence et comparaison avec le nombre de 1 jusqu'à la $n^{\text{ème}}$ case.

Exercice : Enoncé

Distance de Bhattacharrya

- On cherche à calculer la distance de Bhattacharrya entre 2 distributions (discrètes) de probabilité. La distance de Bhattacharyya est donnée par $d(\mathbf{h}_1, \mathbf{h}_2) = -\log\left(\sum_{b=1}^{B}\sqrt{\mathbf{h}_1(b) \times \mathbf{h}_2(b)}\right)$ où $\mathbf{h}_1(b)$ et $\mathbf{h}_2(b)$ sont les valeurs des distributions 1 et 2 en b, avec $b = 1, \ldots, B$.
- Les distributions sont stockées dans des fichiers textes de 2 lignes : la première ligne contient le nombre de cases et la ligne suivante les valeurs de chaque case.

Question

- Écrire une fonction permettant de lire une distribution.
- Implémenter une fonction de calcul de la distance de Bhattacharrya. On renverra -1 si les 2 distributions n'ont pas la même longueur.

UPMC SORBONNE UNIVERSITÉS

Exercice : Analyse

Lecture d'une distribution

- Entrée : nom du fichier à lire
- Résultats :
  - distribution (tableau)
  - taille du tableau

$\Rightarrow$ il faut passer un des 2 résultats comme paramètres de la fonction

Distance de Bhattacharrya

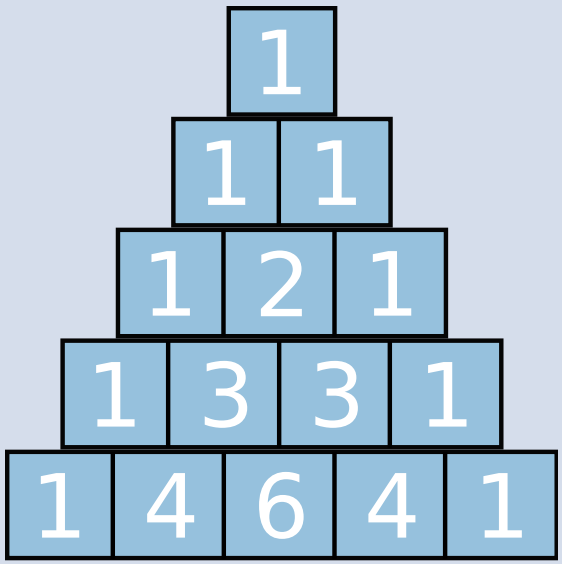
- Entrées :
  - distribution 1 : tableau et taille
  - distribution 2 : tableau et taille
- Résultats : la distance entre les 2 distributions

Exercice : Enoncé

Triangle de Pascal

- On souhaite calculer et stocker dans une matrice les valeurs des coefficients binomiaux

$$C_n^p = \begin{cases} C_{n-1}^{p-1} + C_{n-1}^p & \forall p \in [1; n[\\ 1 \quad \text{si } n = p \text{ ou } p = 0 \end{cases}$$



Question

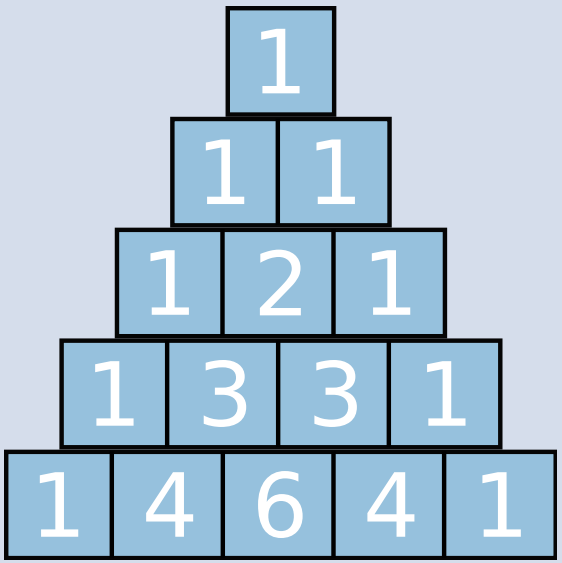
- Implémenter une fonction permettant de calculer le triangle de Pascal

Exercice : Enoncé

Triangle de Pascal

- On souhaite calculer et stocker dans une matrice les valeurs des coefficients binomiaux

$$C_n^p = \begin{cases} C_{n-1}^{p-1} + C_{n-1}^{p} & \forall p \in [1; n[\\ 1 \quad \text{si } n = p \text{ ou } p = 0 \end{cases}$$



Entrées et Sorties

- Données : nombre de lignes du triangle N
- Résultat : triangle de Pascal (tableau 2D)