Licence STS

Université Claude Bernard Lyon I

LIFAP1 : ALGORITHMIQUE ET PROGRAMMATION IMPÉRATIVE, INITIATION

COURS 5 : Les Tableaux

PLAN DE LA SÉANCE

- Comprendre l’utilité des tableaux
- Apprendre à manipuler des tableaux
  - 1 dimension
  - 2 dimensions
  - Multi-dimensions
- Application au cas particulier des ensembles

UTILITÉ DES TABLEAUX : EXEMPLE

- Calcul d'une moyenne de n notes
- Solution sans tableau
  - Déclarer autant de variables que de notes
  - Écrire la somme de ces n variables
- Implique de connaître au départ le nombre de notes (pour déclarer le bon nombre de variables)
- Notation très lourde (surtout si beaucoup de notes à gérer…)
- **Idée** : rassembler toutes ces variables dans une structure de données particulière : le tableau !!!

TABLEAU : DÉFINITION

- **Structure de données** qui contient une collection d'éléments de **même** type
  - exemple : tableau d’entiers, de réels, de caractères…
- Chaque élément a une position définie dans le tableau : désignée par un **indice**
- L’indice d’un tableau est nécessairement de type **entier**

TABLEAU : DÉFINITION

Un tableau est de **taille fixe**, définie lors de sa déclaration

- Chaque élément est manipulé individuellement
- Pas d'opération de manipulation globale de tableau
  - affichage du contenu du tableau
  - initialisation du tableau
  - …

TABLEAU À UNE DIMENSION : DÉCLARATION

- T : tableau [nbcases] de type
  - T : tableau [10] de entier
  - T désignera un tableau contenant 10 valeurs de type entier
  - Attention : les **indices valides** seront compris entre 0 et 9 inclus
- Indice < nombre d'éléments du tableau !!!
- Chaque entrée (élément) du tableau sera désignée par son indice
  - T[i-1] désignera la $i^{ème}$ case du tableau

TABLEAU : STRUCTURE DE STOCKAGE

- Un tableau permet de stocker différentes informations ayant le **même** type
- Chaque élément est identifié par sa position
- Nombre d'entrées maximal :
  - fixé par la déclaration = taille du tableau
- Nombre d'entrées utilisées :
  - à mémoriser dans une ou plusieurs variables à gérer
- On peut déclarer un tableau de 10 cases et n’en utiliser que 5 → surdimensionnement
- Attention la réciproque n’est pas vraie !!
  Si on déclare 5 cases on ne peut pas accéder à la 7ème

TABLEAU : REMPLISSAGE

- Complet (toutes les cases contiennent une valeur)

3	6	8	2	9	0	4
0	1	2	3	4	5	6

- Partiel : certaines cases sont vides
  - Premières cases seulement sont utilisées

3	6	8	2			
0	1	2	3	4	5	6

  - Cases entre deux indices i et j donnés remplies

		8	2			
0	1	2	3	4	5	6

TABLEAU REMPLI PARTIELLEMENT

- De nombreux algorithmes nécessitent de travailler sur une partie de tableau identifiée par :
  - indice de début (noté p dans la suite)
  - indice de fin (noté q)
- Il faut à tout moment être capable de savoir en quels indices le tableau est rempli

TABLEAU ET SOUS-PROGRAMMES

- Une fonction **ne peut pas** retourner un tableau

- Attention, en C/C++
  - Dans les sous-programmes, les tableaux sont **TOUJOURS** passés en donnée / résultat (par adresse)

  ➔Mais pas de & !!

INITIALISATION D'UN TABLEAU

- Par défaut les tableaux sont "vides" :
  - c'est-à-dire **pas initialisés**
- Il est incorrect d'accéder à une case qui ne contient rien ou n'importe quoi !!!
  - mais l'ordinateur ne vous le dira pas
- Initialisation : donner à chacune des cases du tableau une valeur
- En général on met des **0** partout
- Certains langages acceptent les initialisations des tableaux "en bloc"
  - Cas du C par exemple : int T[10]={0};

INITIALISATION D'UN TABLEAU

procédure initialisation (T : tableau[10] de entier)

préconditions : aucune

donnée/résultat : T

description : met des 0 dans toutes les cases du tableau

variable locale : indice : entier

début

indice ← 0

Tant Que indice < 10 **Faire**

T[indice] ← 0

indice ← indice + 1

Fin Tant Que

fin

INITIALISATION D'UN TABLEAU

procédure initialisation (T : tableau[10] de entier)

préconditions : aucune

donnée/résultat : T

description : met des 0 dans toutes les cases du tableau

variable locale : indice : entier

début

Pour indice **allant de** 0 **à** 9 **par pas de** 1 **faire**

T[indice]← 0

Fin pour

fin

PERMUTATION DE 2 ÉLÉMENTS D'UN TABLEAU

- On connaît les deux indices des cases à permuter notées i et j
- On passe par l'intermédiaire d'une variable tampon de même type que le contenu du tableau
- On effectue la permutation
- Tableau donné et modifié → donnée et résultat

PERMUTATION DE 2 ÉLÉMENTS D'UN TABLEAU

procédure permutation (T : tableau[10] de entier, i : entier, j : entier)

préconditions : $0 \leq i \leq 9$, $0 \leq j \leq 9$

données : i, j

donnée/résultat : T

Description : effectue la permutation de deux éléments dans un tableau

variable locale : tampon : entier

Début

```
tampon ← T[i]
T[i] ← T[j]
T[j] ← tampon
```

fin

RECHERCHE DU PLUS PETIT ÉLÉMENT SUR UNE PARTIE DU TABLEAU (INDICES)

- Pour rechercher le minimum :
  - initialisation : hypothèse que le premier élément (correspondant à l'indice p) est le plus petit du tableau
  - balayage des éléments d'indices **p+1 à q** pour chercher **éventuellement** un élément plus petit qui **deviendra** le minimum "courant"
  - en fin de balayage, le plus petit élément est trouvé

RECHERCHE DU MINIMUM D'UN TABLEAU : ALGORITHME

```
fonction minimum( T : tableau[100] de entier, p : entier, q : entier) : entier
    Données : T, p, q
    Préconditions :100>q>=p >= 0
    Description : retourne le minimum d'un tableau
    Variable locale : i : entier,m : entier
        Début
        m ← T[p]
        i ← p + 1
        Tant Que i <= q Faire
            Si T [i] < m Alors
                m ← T[i]
            Fin Si
            i ← i + 1
        Fin Tant Que
        Retourner m
    Fin
```

TABLEAU À 2 DIMENSIONS

- **Déclaration :**

T : tableau [10] [5] d'entiers

- T sera un tableau
  de 10 lignes
  et 5 colonnes
- Accès :
  - T[i-1][j-1]
  - désigne la case
    à la $i^{ème}$ ligne
    et $j^{ème}$ colonne

T[0][0]	T[0][1]			
T[1][0]				
	T[2][1]			
			T[6][3]	

TABLEAU À 2 DIMENSIONS : UTILITÉ

- Modélisation de la notion mathématique de matrice
- Modélisation d'une grille :
  - Bataille navale
  - Tétris
- Modéliser une surface ou un plan

INITIALISATION

procédure initialisationA0 (T : tableau[10][10] de entier)

préconditions : aucune

donnée/résultat : T

description : met des 0 dans toutes les cases du tableau 2D

variable locale : i : entier, j : entier

```
début
    Pour i allant de 0 à 9 par pas de 1 faire
        Pour j allant de 0 à 9 pas de 1 faire
            T[i][j] ← 0
        Fin Pour
    Fin Pour
fin
```

LA MATRICE IDENTITÉ

- Matrice carrée : tableau de taille n*n
- Des 0 partout sauf sur la diagonale : si i=j alors on met un 1
- Algorithme de remplissage
  - On initialise dans un premier temps avec que des 0 (initialisationA0)
  - On met les 1 sur la diagonale T[i][i]

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

LA MATRICE IDENTITÉ : “POUR”

procédure identité (T : tableau[10][10] de entier)

préconditions : aucune

donnée/résultat : T

description : met des 1 sur la diagonale du tableau

variable locale : i : entier

début

initialisationA0(T)

Pour i **allant de** 0 à 9 **par pas de** 1 **faire**

T[i][i] $\leftarrow$ 1

Fin Pour

Fin

REMARQUES, EXTENSION ND

- Comme pour les tableaux 1 dimension, les nombres de lignes et de colonnes effectivement utilisées peuvent être passés en paramètres :
  - taille dans chaque dimension
  - indices début et fin dans chaque dimension (bloc)
  - ➢ Utilisation partielle de la matrice
- Nombre de dimensions aussi grand que l'on veut :
  - T3 : tableau [N][M][O] de truc
    - à 3 dimensions
  - Limitation dues :
    - Représentation graphique et "visuelle" difficile pour programmeur
    - Manipulation des indices

STRUCTURE ABSTRAITE : L'ENSEMBLE

- Application directe des tableaux
- Objet mathématique

- Restriction à un ensemble fini
  - Chaque élément est unique
  - Une valeur appartient ou n'appartient pas à un ensemble
  - Opérations sur les ensembles :
    - Union / intersection / différence
  - Ordre partiel : relation d'inclusion

ENSEMBLE VIA L'UTILISATION D'UN TABLEAU

- Déclaration du tableau :
  - Dimension = cardinalité maximale de l'ensemble

- Si toutes les positions du tableau ne sont pas significatives, il faut mémoriser celles qui contiennent des données valides :
  - généralement placées en début de tableau
  - une variable indique le nombre de positions valides à partir du premier indice
    (n éléments occupent les indices compris entre 0 et n-1)

TABLEAU DES 10 PREMIÈRES VALEURS DE LA FACTORIELLE

- Conditions d'ensemble vérifiées ?
  - ✓ Ensemble fini : 10 valeurs uniquement
  - ✓ Valeurs uniques : les valeurs de la factorielle pour n de 1 à 10 sont bien toutes différentes

- On peut utiliser ce concept mathématique pour formaliser le problème
- Définition d'un tableau contenant ces valeurs

TABLEAU DES 10 PREMIÈRES VALEURS DE LA FACTORIELLE

- Déclaration :
  - Fact10 : tableau [10] de entier

- Les valeurs contenues dans le tableau sont indéterminées
  - **Procédure d'initialisation**

- <u>Attention</u> : par définition, les tableaux seront passés en ***Données/Résultats***
  - c'est à dire que les <u>modifications</u> des entrées du tableaux seront <u>conservées après l'exécution</u> de la fonction ou de la procédure

RELATION D'APPARTENANCE

- Test booléen : renvoie vrai ou faux
- Répond à la question : la valeur x appartient-elle à Fact10 ?

- Pour répondre à cette question, la valeur x sera comparée aux éléments contenus dans le tableau Fact10 jusqu'à :
  - soit trouver un élément dont la valeur est égale à x,
    <u>la valeur x appartient à Fact10</u>
  - soit tous les éléments ont été comparés à x
    et aucun n'est égal,
    <u>la valeur x n'appartient pas à Fact10</u>

RELATION D'APPARTENANCE

- Amélioration : on s'arrête dès qu'on trouve une valeur supérieure à celle recherchée
  - Car les valeurs de la factorielle sont rangées dans l'ordre croissant dans le tableau :
    factorielle(n)<factorielle(n+1) pour tout n
  - Le tableau est donc trié
- Définir la relation d'appartenance revient donc à chercher l'élément dans le tableau

LA RELATION D'APPARTENANCE : ALGORITHME

```
fonction appartientAFact10( Fact10 : tableau[10] de entier, x : entier) : booléen
    Données : x
    Données / Résultat : Fact10
    Préconditions : aucune
    description : teste si l’entier x appartient au tableau
    Variable locale : i : entier
        début
        i ← 0
        Tant Que i < 10 Faire
            Si Fact10[i] = x Alors
                Retourner Vrai
            Fin Si
            i ← i + 1
        Fin Tant Que
        Retourner Faux
    Fin
```

LA RELATION D'APPARTENANCE : ALGORITHME

- Ici, on ne réécrit pas l'algorithme avec une boucle POUR à la place du TANT QUE :
  - Le nombre maximum d'itérations est connu (10)
  - Mais il est possible de sortir avant la fin si on trouve l'élément

- Variante : on sort de la boucle dès qu'on dépasse la valeur recherchée
  - Condition supplémentaire dans le "tant que"

LA RELATION D'APPARTENANCE : ALGORITHME

```
fonction appartientAFact10( Fact10 : tableau[10] de entier, x : entier) : booléen
    Données : x
    Données / résultat : Fact10
    Préconditions : aucune
    Description : teste si l'entier x appartient au tableau
    Variable locale : i : entier
        début
        i ← 0
        Tant Que (i < 10) et (Fact10[i]<= x) Faire
            Si Fact10[i] = x Alors
                Retourner Vrai
            Fin Si
            i ← i + 1
        Fin Tant Que
        Retourner Faux
    Fin
```

EXTENSION DU PROBLÈME

- Si on voulait maintenant les 15 premières valeurs de la factorielle
- Faut-il réécrire la fonction d'appartenance ?
  - Seule la taille du tableau change,
    - dans la déclaration
    - dans le test d'arrêt de la boucle

- Paramétrer !
  - La taille du tableau si balayage complet
  - Indices de début et de fin, pour travailler sur une partie du tableau

APPARTENANCE PARAMÉTRÉE

fonction appartientA (T : tableau[100] de entier, n : entier, x : entier) : booléen

Données : x, n (n: nombre de cases occupées dans le tableau)

Données /résultat : T

Préconditions : 100 > n > 0

Description : teste si l'entier x appartient au tableau

Variable locale : i : entier

Début

i ← 0

Tant Que (i < n) et (T[i]<= x) **Faire**

Si T [i] = x **Alors**

Retourner Vrai

Fin Si

i ← i + 1

Fin Tant Que

Retourner Faux

Fin

LES TABLEAUX EN C

- Déclaration :
  type T[dimension]; //tableau à 1 dimension
  type T[ligne][colonne]; //tableau à 2 dimensions

- Opérations sur le tableau :
  - Aucune à part initialisation (limitation du C/C++)
- Opérations sur un élément :
  - Un élément T[i] est une variable,
    les mêmes opérations sont disponibles.
- Utilisation comme paramètre :
  - Identique à la déclaration

UTILISATION DES TABLEAUX EN C : REMPLISSAGE

```
int main(void)
{
  int tableau[10];
  int i;
      /* remplir le tableau */
  i= 0;
  while(i < 10)
  {
      tableau[i]= i;
      i= i+1;
  }
  return 0;
}
```

Déclaration du tableau de 10 entiers

Remplissage de la case i avec comme valeur celle de son indice

EXEMPLE : AFFICHAGE D'UN TABLEAU

```
void affiche(int T[10])
{
    int i;

    i= 0;
    while(i < 10)
    {
            cout << T[i];
            i= i+1;
    }
}


int main(void)
{
    int tableau[10];
    ...
    // remplir le tableau

    affiche(tableau);
    return 0;
}
```

LIMITATIONS DU C/C++

- C/C++ ne permet ni de renvoyer plusieurs valeurs,
  ni de renvoyer un tableau
  ➔ uniquement des types de retour simples (entier, réel, booléen)

- Transformer les fonctions concernées (plusieurs résultats ou tableau) en procédures et utiliser des paramètres résultats supplémentaires. (cf. CM4)

CONCLUSION

- Structure de données tableau
  - 1 dimension
  - N dimensions
  - De n'importe quoi
- Notion d'ensemble mathématique modélisé dans un tableau
- Algorithmes de bases