

Rappel

1

Commenter un ensemble de lignes

```
/* .... Blabla ....  
*/
```

Commenter une ligne

```
//
```

Remarque : les commentaires

2

- 3 types de commentaires :
 - En début de programme
 - Pour décrire ce que fait le programme ou l'ensemble de fonctions définies dans le fichier
 - Auteur(s), version, historique ...
 - Surtout dans le fichier contenant le main et les fichiers .h (header) pour décrire l'ensemble des fonctions définies
 - Pour chaque fonction
 - Décrire ce que fait la fonction
 - Quels sont les paramètres utilisés, sont-ils modifiés ?, ...
 - La valeur retournée par la fonction (si y'en a une)
 - Au niveau du code
 - Pour décrire ce que fait un bloc d'instructions ou une suite d'instruction lorsque c'est nécessaire

Les commentaires : exemple

3

```
// ----- C++ header File -----
// Author   : Gildas Belay (FT R&D)
// version  : 0.8 {
//
// -----
//
// TestTri.cpp : programme de test pour le tri
//
// //////////////////////////////////////
//
// ....
//
// **
// * Tri le tableau d'entiers tab par ordre croissant en utilisant
// * la methode par selection
// *
// * in/out : tab   tableau d'entier non trié
// * in   : taille nombre d'element dans le tableau
// * return : true  si la fonction s'est bien deroulee
// */
bool triSelection(int *tab, int taille) {
    ...
}
```

Allocation mémoire dynamique

4

Jusqu'à maintenant on a vu :

```
int toto[10]; ou
#define taille 10
int toto[taille];
```

La taille des tableaux est définie lors de la compilation et ne peut être changée !!!

Comment « allouer » un tableau dont on ne connaît pas la taille lors de la compilation (taille en fonction d'un paramètre utilisateur) ??

⇒ allocation dynamique de tableau

Syntaxe :

```
vartype *varnam = new vartype[taille];
```

Libérer la mémoire :

```
delete varnam;
```

Exemples:

```
int *toto;
```

```
toto = new int[50];
```

```
float * titi = new float[nbelem]; (nbelem = var. entière)
```

```
delete titi;
```

Les algorithmes de tri

Sommaire

7

- La fonction de permutation : echanger
- Les tris élémentaires
 - Le tri par création
 - Le tri sélection
 - Le tri par insertion
- Les tris rapides
 - Le tri par fusion
 - Le tri « casiers »
- Comparaison de performances
- Conclusion

Rq: pour simplifier les algos, nous utiliserons uniquement des tableaux d'entiers à tirer.

La fonction de permutation

8

- Pour trier les valeurs d'un tableau, il est nécessaire de permuter des valeurs contenues dans les différentes cases du tableau
- Pour cela, une fonction de permutation, qui s'appelle « echanger », doit être écrite.
- Arguments :
 - Un tableau
 - Deux entiers i et j
- Fonctionnement :
 - La fonction récupère le contenu de la $i^{\text{ème}}$ case du tableau, affecte à cette case la valeur contenue dans la $j^{\text{ème}}$ case, puis affecte à la $j^{\text{ème}}$ case la valeur initiale de la case i.

```
void echanger(int tab[ ], int i, int j);
```

Fonction echanger

```
// Permute 2 éléments d'un tableau
void echanger(int tab[ ], int i, int j) {
    int memoire;
    memoire = tab[i];
    tab[i] = tab[j];
    tab[j] = memoire;
}
```

Le tri par création

Le principe de ce tri est de créer un tableau vide de même taille que le tableau à trier, puis de chercher dans le tableau à trier le plus petit élément afin de le placer en première position du tableau nouvellement créé.

Ensuite, l'algorithme recherche successivement les éléments suivants afin de les placer, à la suite, dans le nouveau tableau. L'algorithme se termine lorsque tous les éléments du tableau ont été ajoutés.

5	3	1	2	6	4
---	---	---	---	---	---

1	?	?	?	?	?
1	2	?	?	?	?
1	2	3	?	?	?
1	2	3	4	?	?
1	2	3	4	5	?
1	2	3	4	5	6

1. Définir une structure permettant de simplifier cette méthode de tri (quels sont les éléments déjà triés ?)
2. Écrire une procédure permettant d'initialiser un tableau de ce type à partir d'une série d'entiers
3. Écrire la procédure tri_creation retournant un tableau trié

11

```

#define NBELEM 6

// Variables globales
int tabval[NBELEM] = { 5, 3, 1, 2, 6, 4};
bool isdone[NBELEM] = {false, false, false, false, false, false};

int triCreation_get_index_min() {
// retourne l'index de l'element le + petit non encore trié
// -1 si pas trouvé (<=> tous les elements sont triés)
int res = -1;

// trouver le 1er index (s'il existe)
int i = 0;
bool found = false;
while ( (i<NBELEM) && (!found) ) {
    if (!isdone[i]) {
        found = true;
        res = i;
    }
    i++;
}

if (!found) return res;

// chercher dans le reste du tableau s'il existe une val <
while (i<NBELEM) {
    if ( !isdone[i] && (tabval[i] < tabval[res]) )
        res = i;
    i++;
}
return (res);
}

void triCreation(int input[NBELEM], int output[NBELEM]) {
// Tri le tableau input -> output
int i;

for (i=0; i<NBELEM; i++) {
    int next_index = triCreation_get_index_min();
    if (next_index != -1) {
        isdone[next_index] = true;
        output[i] = input[next_index];
    }
}

void inittab(tabtri &t) {
for (int i=0; i<NBELEM; i++) {
    cin >> t.tabval[i];
    t.isdone[i] = false;
}
}

```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

12

Tri sélection (complexité $O(n^2)$)

- Ce tri opère en place, i.e le tableau t va être remplacé par son contenu trié.
- Le tri consiste à chercher le + petit élément de $t[0..n]$, soit $t[m]$. A la fin du calcul, cette valeur devra occuper la case 0 de t.
- \Rightarrow d'où l'idée de permuter les valeurs de $t[0]$ et $t[m]$. Il ne reste ensuite plus qu'à trier le tableau $t[1..n]$

Rq: il suffit d'arrêter la boucle à $n-2$ au lieu de $n-1$ puisque le tableau $t[n-1..n]$ sera automatiquement trié.

int[] triSelection(int [] t, int nbelem);

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

13

```

// Tri par selection
void triSelection(int t[], int nbelem) {
    for (int i = 0; i < nbelem-1; i++){
        // invariant: t[0..i] contient les i plus petits
        // éléments du tableau de départ

        // recherche de l'indice du minimum de t[i..n[
        int m = i;
        for (int j = i+1; j < nbelem; j++){
            if (t[j] < t[m]) m = j;

        // on échange t[i] et t[m]
        echanger(t, i, m);
    }
}

```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

14

Tri par insertion (complexité $O(n^2)$)

Le tri est celui du joueur de cartes qui veut trier son jeu.

On prend en main sa première carte ($t[0]$), puis on considère la deuxième ($t[1]$) et on la met devant ou derrière la première, en fonction de sa valeur.

Après avoir classé ainsi les $i-1$ premières cartes, on cherche la place de la i -ième, on décale alors les cartes pour insérer la nouvelle carte.

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

15

Tableau à trier	3	5	7	3	4	6
1 ^{ère} valeur	3					
Pour insérer le 5 (5 > 3) .. Idem pour 7	3	5	7			
Pour insérer le 3, on doit décaler les 5 et 7	3		5	7		
Puis insérer le 3	3	3	5	7		
Finalemt on obtient	3	3	4	5	6	7

University of Geneva
www.miralab.chLCI
Introduction au langage C

16

```

// Tri par insertion
void trinsertion(int t[], int nbelem) {
    for (int i = 1; i < nbelem; i++) {
        // t[0..i-1] est déjà trié

        int j = i;
        // recherche la place de t[i] dans t[0..i-1]
        while ( ( j > 0 ) && (t[j-1] > t[j]) ) {
            j--;
        }

        // si j = 0, alors t[j] <= t[0]
        // si j > 0, alors t[j] > t[j-1]
        // dans tous les cas, on pousse t[j..i-1] vers la droite

        int tmp = t[j];
        for (int k = i; k > j; k--)
            t[k] = t[k-1];

        t[j] = tmp;
    }
}

// Tri par insertion (avec nouveau tableau)
int* trinsertion_creation(int t[], int nbelem) {
    int *tres = new int[nbelem];
    // allocation memoire dynamique d'un tableau
    tres[0] = t[0];

    for (int i = 1; i < nbelem; i++) {
        // tres[0..i-1] est déjà trié

        int j = i;
        // recherche la place de t[i] dans tres[0..i-1]
        while ( ( j > 0 ) && (tres[j-1] > t[j]) ) {
            j--;
        }

        // si j = 0, alors t[j] <= t[0]
        // si j > 0, alors t[j] > t[j-1]
        // dans tous les cas, on pousse t[j..i-1] vers la droite

        int tmp = t[j];
        for (int k = i; k > j; k--)
            tres[k] = tres[k-1];

        tres[j] = tmp;
    }

    return (tres);
}

```

University of Geneva
www.miralab.chLCI
Introduction au langage C

Tri rapide : tri par fusion ($O(n \log n)$)

Le tri fusion est construit suivant la stratégie "diviser pour régner", en anglais "divide and conquer". Le principe de base de la stratégie "diviser pour régner" est que pour résoudre un gros problème, il est souvent plus facile de le diviser en petits problèmes élémentaires. Une fois chaque petit problème résolu, il n'y a plus qu'à combiner les différentes solutions pour résoudre le problème global.

La méthode "diviser pour régner" est tout à fait applicable au problème de tri : plutôt que de trier le tableau complet, il est préférable de trier deux sous tableaux de taille égale, puis de fusionner les résultats.

L'algorithme proposé ici est récursif. En effet, les deux sous tableaux seront eux même triés à l'aide de l'algorithme de tri fusion. Un tableau ne comportant qu'un seul élément sera considéré comme trié : c'est la condition sine qua non sans laquelle l'algorithme n'aurait pas de conditions d'arrêt.

Etapas de l'algorithme :

- Division de l'ensemble de valeurs en deux parties
- Tri de chacun des deux ensembles
- Fusion des deux ensembles

Tableau	Commentaire : ce qui est fait pour passer à la ligne suivante
6 3 0 9 1 7 8 2 5 4	Division du tableau en deux parties
6 3 0 9 1 7 8 2 5 4	Division de chaque sous tableau en deux parties
6 3 0 9 1 7 8 2 5 4	Division de chaque sous tableau en deux parties
6 3 0 9 1 7 8 2 5 4	Division des sous tableau bleu et rouge en deux parties, fusion des tableaux vert-rose
6 3 0 1 9 7 8 2 4 5	Fusion des sous tableaux bleu-rose et rouge-rose
3 6 0 1 9 7 8 2 4 5	Fusion des sous tableaux bleu-jaune et rouge-jaune
0 3 6 1 9 2 7 8 4 5	Fusion des sous tableaux bleu-vert et rouge-vert
0 1 3 6 9 2 4 5 7 8	Fusion des deux tableaux
0 1 2 3 4 5 6 7 8 9	Le tableau est trié, l'algorithme est terminé

19

```

void fusion(int tableau[],int deb1,int fin1,int fin2) {
    int *table1;
    int deb2=fin1+1;
    int compt1=deb1;
    int compt2=deb2;
    int i;

    table1 = new int[fin1-deb1+1];

    //on recopie les éléments du début du tableau
    for(i=deb1;i<=fin1;i++) {
        table1[i-deb1]=tableau[i];
    }

    for(i=deb1;i<=fin2;i++) {
        if (compt1==deb2) {
            //c'est que tous les éléments du premier tableau ont été utilisés
            break; //tous les éléments ont donc été classés
        }
        else if (compt2==(fin2+1)) {
            //c'est que tous les éléments du second tableau ont été utilisés
            tableau[i]=table1[compt1-deb1]; //on ajoute les éléments restants du premier tableau
            compt1++;
        }
        else if (table1[compt1-deb1]<tableau[compt2]) {
            tableau[i]=table1[compt1-deb1]; //on ajoute un élément du premier tableau
            compt1++;
        }
        else {
            tableau[i]=tableau[compt2]; //on ajoute un élément du second tableau
            compt2++;
        }
    }

    delete table1;
}

void tri_fusion_bis(int tableau[],int deb,int fin) {
    if (deb != fin) {
        int milieu = (fin+deb)/2;
        tri_fusion_bis(tableau, deb, milieu); //
        division
        tri_fusion_bis(tableau, milieu+1, fin); //
        division
        fusion(tableau, deb, milieu, fin); // fusion
        des elements
    }
}

void tri_fusion(int tableau[],int nbelem) {
    if (nbelem>0) {
        tri_fusion_bis(tableau, 0, nbelem-1);
    }
}

```

20

Tri par casier

Le tri par casiers est une méthode de tri très spéciale car elle ne s'applique qu'au tri de valeurs entières incluses dans un ensemble par trop grand. Cette contrainte réduit énormément la portée des utilisations de ce tri, mais quand ce tri est implémentable, il est d'une redoutable efficacité. Cela est dû au fait que c'est la seule méthode de tri qui ne nécessite aucune comparaison entre les différents éléments.

Son principe est le suivant :

- Il faut rechercher dans l'ensemble d'éléments le plus petit élément u et le plus grand élément v .
- Dans un deuxième temps, un tableau de taille $(v-u+1)$ dont toutes les valeurs sont initialisées à 0 est construit.
- Ensuite, l'ensemble des valeurs à classer est parcouru, et pour chaque valeur x , la case n° $(x-u)$ est incrémentée.
- Enfin, le tableau est parcouru et l'ensemble des éléments est recomposé en créant pour chaque case, autant de valeur qu'indiqué par son contenu.

21

N° de case	0	1	2	3	4	5	6
Élément x correspondant	-1	0	1	2	3	4	5
Tableau initial	0	0	0	0	0	0	0
É l é m e n t x	4	0	0	0	0	1	0
	0	0	1	0	0	1	0
	-1	1	1	0	0	1	0
	2	1	1	0	1	0	1
	0	1	2	0	1	0	1
	0	1	3	0	1	0	1
	2	1	3	0	2	0	1
	5	1	3	0	2	0	1
	-1	2	3	0	2	0	1
	2	2	3	0	3	0	1
	0	2	4	0	3	0	1
	4	2	4	0	3	0	2
	2	2	4	0	4	0	2

D'après le tableau on a	Soit l'ensemble suivant :
2 fois -1	-1 -1
4 fois 0	-1 -1 0 0 0 0
0 fois 1	-1 -1 0 0 0 0
4 fois 2	-1 -1 0 0 0 0 2 2 2 2
0 fois 3	-1 -1 0 0 0 0 2 2 2 2
2 fois 4	-1 -1 0 0 0 0 2 2 2 2 4 4
1 fois 5	-1 -1 0 0 0 0 2 2 2 2 4 4 5

22

Implémenter l'algorithme du tri à casier !

```
void triCasier(int tableau[],int nbelem)
```

23

```

void triCasier(int tableau[], int nbelem) {
    int min, max, i, longueur2, compt;
    int *tableau2;

    //recherche des valeurs min et max
    min = tableau[0];
    max = min;

    for(i=1; i<nbelem; i++) {
        if (tableau[i]<min) min=tableau[i];
        if (tableau[i]>max) max=tableau[i];
    }

    //on construit le tableau intermédiaire
    longueur2 = max - min + 1;
    tableau2 = new int[longueur2];

    //on initialise le tableau à 0
    for(i=0; i<longueur2; i++)
        tableau2[i] = 0;

    //on compte le nombre d'occurrences de chaque
    entier
    for(i=0; i<nbelem; i++)
        tableau2[tableau[i] - min]++;

    compt=0;
    for(i=0; i<longueur2; i++) {
        while (tableau2[i]>0) {
            tableau[compt] = min+i;
            tableau2[i]--;
            compt++;
        }
    }

    delete (tableau2);
}

```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

24

Comparaison de performances

Algorithme	Pire des cas	En moyenne	Meilleur des cas
Tri Bulle	$O(n^2)$		
Tri Bulle Optimisé	$O(n^2)$		
Tri par Selection	$O(1/2 * n^2)$	$O(1/2 * n^2)$	$O(1/2 * n^2)$
Tri par Insertion	$O(1/2 * n^2)$	$O(1/4 * n^2)$	
Tri Shell	$O(n^2)$	$O(n^2)$	
Tri Fusion	$O(n * \lg(n))$	$O(n * \lg(n))$	$O(n * \lg(n))$
Tri Rapide	$O(n * \lg(n))$	$O(n * \lg(n))$	$O(n^2)$
Tri Casier	$O(n + m)$	$O(n + m)$	$O(n + m)$
Tri par Création	$O(n^2)$	$O(n^2)$	$O(n^2)$
Tri par Arbre Binaire de Recherche	$O(n * \lg(n))$	$O(n * \lg(n))$	$O(n * \lg(n))$

n = nombre d'éléments à trier
m = cardinal des éléments à trier

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Divers algorithmes

University of Geneva
www.miralab.ch

Recherche du plus petit élément

26

Trouver le plus petit élément d'un tableau

```
int plusPetit(int t[], int nbelem);
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

```
// retourne la + petite valeur de t
int plusPetit_valeur(int t[], int nbelem) {
    int k = 0; // par défaut t[0] est le plus petit element
    for(int i = 1; i < nbelem; i++) {
        // invariant : k est l'indice du plus petit élément de x[0..i-1]
        if(t[i] < t[k]) k = i;
    }
    return t[k];
}

// retourne l'indice du + petit element de t
int plusPetit_indice(int t[], int nbelem) {
    int k = 0; // par défaut t[0] est le plus petit element
    for(int i = 1; i < nbelem; i++) {
        // invariant : k est l'indice du plus petit élément de x[0..i-1]
        if(t[i] < t[k]) k = i;
    }
    return k;
}
```