

COMMENT TRIER DES DONNEES EN C++

1 UTILISATION DE LA FONCTION QSORT

Cette fonction effectue un tri rapide (Tri Quick) sur un tableau de données. C'est à l'opérateur de définir une fonction de comparaison afin de déterminer la manière dont les données sont triées.

1.1 Tri d'un tableau d'entier

```
// Tri d'un tableau d'entiers en utilisant la fonction qsort
```

```
#include <iostream>
```

```
#include <conio.h>
```

```
using namespace std ;
```

```
const int N=10;
```

```
// Déclaration de la fonction de comparaison = son prototype
```

```
int compare(const void * , const void * );
```

```
// Codage de la fonction de comparaison
```

```
int compare(const void * arg1, const void * arg2 )  
{
```

```
    int nb1 , nb2;
```

```
    nb1 = * (int *) arg1 ;
```

```
    nb2 = * (int *) arg2 ;
```

```
    if (nb1 > nb2 )
```

```
        return 1;
```

```
    else if (nb1 < nb2)
```

```
        return -1 ;
```

```
    else return 0;
```

```
}
```

```
// Fonction principale
```

```
int main( )
```

```
{
```

```
    int tab[N] = {4 , 2 , 7 , 1 , 0 , 8 , 12 , 2 , 3 , 6 };
```

```
    qsort(tab , N , sizeof(int) , compare);
```

```
    // Affichage du tableau trie
```

```
    int i;
```

```
    for (i=0 ; i<N ; i++)
```

```
        cout << tab[i] << "\t";
```

```
    _getch();
```

```
    return 0 ;
```

```
}
```

arg1 et arg2 sont deux pointeurs sur les éléments à comparer

// (int *) permet de forcer le pointeur à devenir un pointeur sur un entier

// 0 si les deux nombres sont égaux

Appel de la fonction qsort

- 1^{er} argument = tableau à trier
- 2^{ème} argument = Nombre de valeurs
- 3^{ème} argument = taille en octets d'une valeur
- 4^{ème} argument = nom de la fonction de comparaison

Exercice : Modifier ce programme pour trier les valeurs par ordre décroissant

1.2 Tri d'un tableau de chaînes de caractères

```
#include <conio.h>
#include <iostream>
#include <stdio.h>

using namespace std;

//----- Définition de la fonction de comparaison -----
int compare( const void *arg1, const void *arg2 );

//***** Fonction principale *****
int main()
{
    char **pMots;           // pMots est l'adresse d'une adresse
    char chaine[80+1];
    unsigned NbMots;
    unsigned i;

    cout << "\nRentre le nombre de mots que tu veux trier ";
    cin >> NbMots;

    //---- Réserve d'une zone de NbMots pointeurs en mémoire ----
    pMots = new char * [NbMots];

    //----- Saisie des mots -----
    for (i=0 ; i<NbMots ; i++)
    {
        cout << "\nRentre le mot " << i << " " << flush;
        gets(chaine);

        // Réserve d'une zone mémoire pour stocker chaque mot
        pMots[i] = new char [strlen(chaine)+1];

        // stockage dans cette zone du mot saisi
        strcpy(pMots[i], chaine);
    }

    //----- Tri des mots. En fait ,on trie des pointeurs-----
    qsort( (void *)pMots ,NbMots, sizeof( char * ), compare );

    //----- Affichage -----
    for( i = 0; i <NbMots; i++)    cout << pMots[i] << " " << flush;

    //----- Libération de la mémoire allouée dynamiquement -----
    for (i=0 ; i<NbMots ; i++) delete[] pMots[i];    // d'abord chaque mot
    delete[] pMots;                                // puis le tableau d'adresses

    _getch(); return 0 ;
}

//*****
// Fonction de comparaison de deux mots
//*****
int compare( const void *arg1, const void *arg2 )
{
    return strcmp( *( char** ) arg1, *( char** ) arg2 );
}
```

Exemple : on saisit 4 mots (NbMots= 4)

Les mots saisis sont : "prune", "cerise", "pomme", "poire"

Données	Adresses
pMots[0]=0x00780510	pMots = 0x0078050
pMots[1]=0x007804e0	
pMots[2]=0x007804a0	arg1 (situation à un moment donné)
pMots[3]=0x00780460	arg2 (situation à un moment donné)
'p'	0x00780460
'o'	
'i'	
'r'	
'e'	
'\0'	
'p'	0x007804a0
'o'	
'm'	
'm'	
'e'	
'\0'	
'c'	0x007804e0
'e'	
'r'	
'i'	
's'	
'e'	
'\0'	
'p'	0x00780510
'r'	
'u'	
'n'	
'e'	
'\0'	

En fait, ce programme trie le tableau de pointeurs (pMots[0] à pMots[3]) en fonction du critère de comparaison fourni.

Après le tri , on a pMots[0]=0x007804e0, pMots[1]=0x00780460,
pMots[2]=0x007804a0 et pMots[3]=0x00780510

Exercice : Modifier ce programme pour trier les caractères des chaînes avant de trier les chaînes elles-même ➔ ceers – eiopr – emmop - enpru

2 UTILISATION DE LA BIBLIOTHEQUE STL

La bibliothèque STL permet une grande souplesse pour manipuler les tableaux (vector). Il est également possible d'appliquer un algorithme de tri sur les vecteurs. Avant de voir les tris, revoyons les concepts élémentaires de manipulation des vecteurs.

2.1 Exemple de manipulation de vecteurs

```
#include <iostream>
#include <conio.h>
#include <vector>
```

Inclure <vector> pour utiliser les vecteurs

```
using namespace std ;
```

```
int main( )
{
```

```
    vector<double> TabReels; // Au début le tableau est vide
```

```
    // Ajout de valeurs à la fin du tableau
```

```
    TabReels.push_back(7.5);
    TabReels.push_back(2.25);
    TabReels.push_back(17.3);
    TabReels.push_back(24.6);
    TabReels.push_back(1.125);
```

```
    // Ajout d'une valeur à une position précise, par exemple en 3eme position
```

```
    TabReels.insert(TabReels.begin() + 2 , 14.44);
```

```
    // Affichage du tableau
```

```
    unsigned i;
    cout << "\nAffichage initial du tableau : " << flush ;
    for (i = 0 ; i < TabReels.size() ; i++)
        cout << TabReels[i] << " " << flush;
```


```
    // Suppression d'un élément, par exemple le 4eme
```

```
    TabReels.erase(TabReels.begin() + 3 );
    cout << "\nAffichage apres suppression de la 4eme valeur: " << flush ;
    for (i = 0 ; i < TabReels.size() ; i++)
        cout << TabReels[i] << " " << flush;
```

```
    // Vidage complet du tableau
```

```
    TabReels.clear();
    cout << "\nAffichage apres vidage complet " << flush ;
    for (i = 0 ; i < TabReels.size() ; i++)
        cout << TabReels[i] << " " << flush;
```

```
    _getch();
    return 0 ;
}
```



```
c:\Mes documents\Travail\Cours_07_08\STSH\TP\Tri\Debug\Tri.exe
Affichage initial du tableau : 7.5 2.25 14.44 17.3 24.6 1.125
Affichage apres suppression de la 4eme valeur: 7.5 2.25 14.44 24.6 1.125
Affichage apres vidage complet
```

2.2 Tri d'un vecteur d'entiers

```
#include <iostream>
#include <conio.h>
#include <vector>
#include <algorithm>
```

```
using namespace std;
```

```
// Fonction de comparaison
```

```
bool compare(int x, int y)
{
    if (x > y) return true;
    else return false;
}
```

```
// Fonction principale
```

```
int main()
{
```

```
    unsigned i;
```

```
    vector<int> tab;
```

```
    tab.push_back(3);
```

```
    tab.push_back(7);
```

```
    tab.push_back(1);
```

```
    tab.push_back(8);
```

```
    tab.push_back(2);
```

```
    cout << "\nAffichage du tableau avant le tri " << endl;
```

```
    for (i=0; i<tab.size(); i++)
        cout << tab[i] << "t";
```

```
    sort(tab.begin(), tab.end());
```

```
    cout << "\nAffichage du tableau trie " << endl;
```

```
    for (i=0; i<tab.size(); i++)
        cout << tab[i] << "t";
```

```
// Si on veut trier autrement, par exemple en sens inverse, il faut ajouter une
```

```
// fonction de comparaison
```

```
sort(tab.begin(), tab.end(), compare);
```

```
cout << "\nAffichage du tableau trie avec la fonction de comparaison fournie" << endl;
```

```
for (i=0; i<tab.size(); i++)
    cout << tab[i] << "t";
```

```
_getch();
```

```
return 0;
```

```
}
```

- Inclure <vector> pour utiliser les vecteurs
- Inclure <algorithm> pour utiliser la fonction sort() de la bibliothèque STL

- tab est vide à sa création
- push_back permet d'ajouter un élément à la fin

tab.begin() est l'adresse du 1^{er} élément
tab.end() est l'adresse du dernier élément

On ajoute en 3^{ème} argument le nom de la fonction de comparaison

2.3 Tri d'un vecteur de chaînes de caractères : peu de différences avec le tri d'entiers

```
// Tri d'un tableau de chaînes avec STL
```

```
#include <iostream>
#include <conio.h>
#include <vector>
#include <string>
#include <algorithm>
```

```
using namespace std ;
```

```
bool compare(string x , string y)
{
    if (x > y) return true;
    else return false ;
}
```

```
int main( )
{
    unsigned i;
    vector<string> tab;

    tab.push_back("POMME");
    tab.push_back("POIRE");
    tab.push_back("PRUNE");
    tab.push_back("CERISE");
    tab.push_back("MIRABELLE");

    cout << "\nAffichage du tableau avant le tri " << endl ;
    for (i=0 ; i<tab.size() ; i++)
        cout << tab[i] << "\t";

    sort(tab.begin() , tab.end() );

    cout << "\nAffichage du tableau trie " << endl ;
    for (i=0 ; i<tab.size() ; i++)
        cout << tab[i] << "\t";

    // Si on veut trier autrement, par exemple en sens inverse, il faut ajouter une
    // fonction de comparaison
    sort(tab.begin() , tab.end() , compare);

    cout << "\nAffichage du tableau trie avec la fonction de comparaison fournie" << endl ;
    for (i=0 ; i<tab.size() ; i++)
        cout << tab[i] << "\t";

    _getch();
    return 0 ;
}
```