

Introduction à la programmation en C++

Chaînes de caractères – Fichiers – Vecteurs

Nicolas Audebert

Vendredi 7 décembre 2018



Rendus de TP et des exercices

Les rendus se font sur [Educnet](#), même en cas de retard. **Pas par mail.**

1. Le code rendu **doit compiler**.
2. Le code rendu doit **être propre** (indentation, noms de variables clairs).
3. Le code rendu doit **être commenté** (réponses aux questions, fonctionnement du code).
4. Rassembler le code dans une seule archive (**.zip, .rar, .tar.gz**, etc.).
5. Le code doit contenir **les noms des deux binômes** le cas échéant.

Un exercice ou un TP rendu en retard ou ne respectant pas une des consignes ci-dessus sera pénalisé.

Rappels

Chaînes de caractères

Fichiers

STL : vector

Problème

Comment passer un argument à une fonction **sans copie** mais en interdisant sa modification ?

Solution

On passe les arguments par référence en spécifiant que l'argument ne doit pas être modifié : **on ajoute le mot clé `const`** .

```
const int N = 1000;
class Vector{
    double t[N];
    ...
};
class Matrix{
    double t[N][N];
    ...
};
```

```
void solve(const Matrix &A,
           const Vector &x, Vector& y)
{...}

...
Matrix M;
Vector a,b;
...
solve(M,a,b);
```

Lorsque l'on passe un objet par référence constante, on ne peut accéder qu'aux méthodes de la classe définies comme **constantes**, *i.e.* qu'on a déclaré comme ne modifiant pas l'objet.

```
const int N = 1000;
class Vector{
    double t[N];
public
    double get(int i);
    void set(int i, double v);
    ...
};
class Matrix{
    double t[N][N];
    ...
};
```

```
void solve(const Matrix &A,
           const Vector &x, Vector& y)
{
    ...
    x.set(10, 8); // ERREUR: x est
                  // non modifiable
    x.get(5); // ERREUR
    y.set(1, 5.6); // OK
}
...
```

Constructeur de copie et opérateur =

Par défaut, le constructeur de copie recopie les champs de **a** dans **b**.

```
void Vecteur::operator=(const Vecteur& o){...}  
Vecteur b = a; // équivalent à Vecteur b(a);
```

Il ne doit pas être confondu avec l'opérateur d'affectation (=).

```
Vecteur Vecteur::operator=(const Vecteur& o){  
    ...  
    return o;  
}
```

```
Vecteur a, b;  
b = a;
```

L'opérateur d'affectation renvoie un objet afin de chaîner les égalités :

```
c = b = a; // équivalent à c = (b = a);
```

Rappels

Chaînes de caractères

Fichiers

STL : vector

Les chaînes de caractères

La bibliothèque standard définit un type pour les chaînes de caractères : `std::string`.

```
using namespace std;
string chaine = "toto"; // Création et affectation
char caractere = chaine[2]; // Récupération du troisième caractère
int longueur = chaine.size(); // Accès à la longueur de la chaîne
```

Toutefois, la `std::string` implémente les chaînes de caractères de façon plus complète qu'un simple tableau.

Chaînes de caractères et tableaux de `char`

Avant les objets, les chaînes de caractères étaient manipulées sous forme de tableaux de `char`, c'est-à-dire des pointeurs de type `char*`. Cette pratique est obsolète et à proscrire.

1. Les opérateurs de comparaison usuels sont redéfinis pour utiliser l'ordre lexicographique sur les chaînes de caractères.

```
"a" < "b"; // --> true
"d" >= "a"; // --> false
"a" < "ab"; // --> true
"A" != "a"; // --> true
"abc" == "abc"; // --> true
"cat" < "caterpillar"; // --> true
```

2. Il est possible de chercher une sous-chaîne dans une chaîne.

```
string s = "Ada Lovelace";
size_t i = s.find('a'); // i : indice de h dans s
size_t j = s.find('a', 4); // j : indice de h dans s à partir de 4

size_t k = s.find("ove"); // k : indice de la sous-chaîne dans s
size_t l = s.find("ove", 4); // l : indice dans s à partir de 4
```

Si la recherche n'aboutit pas, `find` renvoie `string::npos`.

3. L'opérateur + est redéfini et correspond à la concénation de deux chaînes :

```
string a = "le début et";  
string b = "la fin";  
  
string sum = a + " " + b + " !";  
cout << sum << endl;  
// "le début et la fin !"
```

D'autres opérations (cf. polycopié)

4. Extraction de sous-chaînes
5. Interaction avec l'utilisateur
6. Chaînes de caractères au format C

Rappels

Chaînes de caractères

Fichiers

STL : vector

Les fichiers se manipulent avec un objet `stream` qui fonctionne de la même façon que `cout` et `cin`.

```
#include<fstream>
using namespace std;
```

Écriture dans un fichier

```
ofstream file;
file.open("chemin/fichier.txt");
// ou ofstream file("chemin/fichier.txt");

file << "La première ligne, numéro " << 1 << endl;
file << "La deuxième ligne, pas de numéro";
file << ", encore la deuxième";
file << endl; // saut de ligne

file.close(); // ne pas l'oublier !
```

Lecture dans un fichier

```
ifstream file("chemin/fichier.csv");  
// mon fichier contient "12 134.0" sur la première ligne  
int mon_entier;  
double mon_reel;  
file >> mon_entier >> mon_reel;  
g.close();
```

Tester si le fichier est ouvert

```
ofstream file.open("chemin/fichier");  
if(!file.is_open()){  
    cout << "Erreur" << endl;  
} else {...}  
f.close();
```

Autres possibilités avec les fichiers : dans le livre.

Rappels

Chaînes de caractères

Fichiers

STL : vector

Standard Template Library

- ▶ Bibliothèque de fonctions standard de C++
- ▶ Elle contient de nombreux modules :
 - ▶ Structures de données : chaînes de caractères, tableaux, piles...
 - ▶ Algorithmes classiques : tri, $n^{\text{ième}}$ éléments...
 - ▶ Lecture/écriture (console, fichiers, réseau...)

Quelques exemples

Nous avons déjà utilisé la STL, notamment les modules `iostream` et `string`.

La STL propose une implémentation de vecteurs dans la classe `vector`.

- ▶ Utilise les tableaux dynamiques
- ▶ Abstraction de la gestion de la mémoire
- ▶ Expose une interface type tableau

Utilisation :

```
#include <vector>
```

```
using namespace std;
```

La classe **vector** est une classe *template*, c'est-à-dire qu'elle s'adapte au type de ses éléments. À la compilation, la classe est spécialisée pour un type spécifique, par exemple **int** ou `Balle`.

```
// Création d'un vecteur d'éléments de type T
vector<T> tab;

// Exemples :
vector<int> t_int;
vector<double> t_double;
vector<Matrix> t_mat;
vector<float*> t_point;
```

Manipulation des vecteurs de la STL

Les vecteurs de la STL s'utilisent de la même façon que les tableaux statiques ou dynamiques.

```
// Création d'un vecteur d'entiers de 100 éléments
vector<int> t(100);

// L'opérateur [] permet d'accéder aux éléments du vecteur
for(int i=0; i<100; i++){
    t[i] = i*i;
}
cout << t[5] << endl;
```

Les vecteurs disposent d'une méthode **size** permettant de récupérer leur taille :

```
cout << t.size() << endl;
```

- ▶ Création et remplissage :

```
// Création d'un vecteur de 100 réels valant tous 5.6  
vector<double> t2(1000, 5.6);
```

- ▶ Redimensionner un vecteur :

```
// Création d'un vecteur de 100 éléments de type T  
vector<T> t(100);  
// Redimensionnement  
t.resize(1000);
```

Les éléments déjà existants sont conservés, sauf si la taille finale du vecteur est inférieure à la taille initiale, auquel cas les éléments surnuméraires disparaissent.

- ▶ Accéder au premier élément :

```
cout << *t.begin() << endl;
```

`t.begin()` est un **itérateur**, qui fonctionne de manière similaire à un pointeur.

- ▶ Accéder à la fin du vecteur :

```
t.end(); // Attention pointe juste derrière la dernière case
```

Ce n'est pas le dernier élément!

- ▶ Concaténer deux vecteurs :

```
vector<int> t1 (10,2);  
vector<int> t2 (30, 100);  
t2.insert(t2.end(), t1.begin(), t1.end());
```

On ajoute le vecteur **t1** à la fin du vecteur **t2**.

- ▶ Trier un vecteur :

```
#include <algorithm>  
  
...  
  
std::sort(t.begin(), t.end());
```

Serpent

Un serpent qui se déplace et s'allonge tout les x pas de temps.

Tron

Un serpent deux joueurs qui s'allonge à tout les pas de temps.

