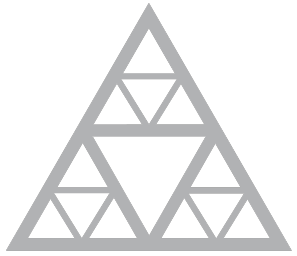
Introduction à la programmation en C++

Chaînes de caractères - Fichiers

Nicolas Audebert

Vendredi 1er décembre 2017



École des Ponts
ParisTech

Avant toute chose

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.).

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

Plan de la séance

Quizz 1

Que fait ce code ?

```
bool paire(Main m){
    bool res = false;
    for(int i=0; i<5; i++){
        for(int j=i; j<5; j++){
            if(m.cartes[i] == m.cartes[j]){
                res = true;
            }
        }
    }
}

bool test = paire();
```

Quizz 1

Que fait ce code ?

```
bool paire(Main m){
    bool res = false;
    for(int i=0; i<5; i++){
        for(int j=i; j<5; j++){
            if(m.cartes[i] == m.cartes[j]){
                res = true;
            }
        }
    }
}

bool test = paire();
```

Il renvoie toujours la même chose, indépendant de `m`. En effet, on a oublié d'ajouter `return res;` à la fin de la fonction…

Quizz 2

Que fait ce code ?

```
Carte c = {VALET, COEUR};
Main m;
...
for(int i=0; i<5; i++){
    if(m.cartes[i] = c){
        cout << "Valet de coeur" << endl;
    }
}
```

Quizz 2

Que fait ce code ?

```
Carte c = {VALET, COEUR};
Main m;
...
for(int i=0; i<5; i++){
    if(m.cartes[i] = c){
        cout << "Valet de coeur" << endl;
    }
}
```

Il change la valeur de toutes les cartes de la main pour les remplacer par le Valet de cœur. En effet, on a confondu l'opérateur d'affectation (=) et l'opérateur d'égalité (==).

Quizz 3

Peut-on raccourcir ce code ?

```
bool ma_fonction(bool test1, bool test2){
    bool result;
    if((test1 && test2) == true){
        result = false;
    } else {
        result = true;
    }
    return result;
}
```

Quizz 3

Peut-on raccourcir ce code ?

```
bool ma_fonction(bool test1, bool test2){
    bool result;
    if((test1 && test2) == true){
        result = false;
    } else {
        result = true;
    }
    return result;
}
```

```
// Oui ! Il faut manipuler directement le booléen.
bool ma_fonction(bool test1, bool test2){
   return !(test1 && test2);
}
```

Quizz 4

Que fait ce code ?

```
int tab[1000];

for(int i=0; i<1000; i++){
    if(i%2 == 0)
        cout << i << " est paire" << endl;
        cout << "Sa moitié est " << i/2 << endl;

    tab[i] = i;
}
```

Quizz 4

Que fait ce code ?

```
int tab[1000];

for(int i=0; i<1000; i++){
    if(i%2 == 0)
        cout << i << " est paire" << endl;
        cout << "Sa moitié est " << i/2 << endl;

    tab[i] = i;
}
```

Si le nombre est pair, il affiche ”n est paire”. Dans tous les cas, il affiche la moitié du nombre. En effet, sans accolades, le `if` ne porte que sur l’instruction suivante. L’indentation ici est trompeuse !

Quizz 5

Que fait ce code ?

```
void remplitCarte(Carte c){
^^I// on remplit une carte aléatoirement
    c.valeur = rand()%13 + 2;
    c.couleur = rand()%4;
}

Carte c;
remplitCarte(c);
cout << c.valeur << ", " << c.couleur << endl;
```

Quizz 5

Que fait ce code ?

```
void remplitCarte(Carte c){
^^I// on remplit une carte aléatoirement
    c.valeur = rand()%13 + 2;
    c.couleur = rand()%4;
}

Carte c;
remplitCarte(c);
cout << c.valeur << ", " << c.couleur << endl;
```

`c` a des valeurs aléatoires. En effet, la fonction `remplitCarte` ne fait que modifier une copie de la carte passée en argument. Il fallait la passer par référence...

Conseils

- Bien **réviser** les bases : passage par valeur, passage par référence, manipulation des booléens, manipulation des tableaux
- **Tester son code** régulièrement et lire attentivement les erreurs de compilation.
- Ne pas hésiter à **afficher le contenu des variables** avec `cout` pour comprendre ce qui se passe.
- **Indenter** correctement son code et ne pas oublier les accolades.
- Mettre des **commentaires**, pour soi et pour le correcteur.

Plan de la séance

Notion de constructeur

L'initialisation d'un objet fait appel au **constructeur** de sa classe.

Définition

Un **constructeur** est une méthode :

- qui **n'a pas** de type de retour,
- qui porte **le même nom** que la classe,
- qui décrit comment initialiser les instances de la classe.

Manipulation

Un constructeur :

- est **systématiquement** appelé à la création d'un objet,
- ne peut pas être appelé **après** la création de l'objet.

Notion de constructeur

Définition

Un **constructeur** est une méthode :

- qui **n'a pas** de type de retour,
- qui porte **le même nom** que la classe,
- qui décrit comment initialiser les instances de la classe.

```
class Point{
    double x,y;
public:
    Point(double valX, double valY);
    ...
}

// Définition du constructeur
Point::Point(double valX, double valY){
    x = valX; y = valY;
}
```

```
...
Point b = {2,3}; // ERREUR
Point c(2,3); // OK
// Cette syntaxe appelle
// le constructeur
```

Le constructeur vide

À la création de l'objet il y a **toujours** un appel à un constructeur.

Lorsqu'aucun constructeur n'est défini par l'utilisateur, le compilateur en créé un par défaut. C'est un **constructeur vide** qui ne prend aucun argument et ne fait que créer les champs de l'objet.

```
class Point{
    double x,y;
public:
    double get(double& x, double& y);
    void set(double valX, double valY);
};
...
Point a;// Appel au constructeur par défaut
```

Référence constantes

Solution

On souhaite passer les objets 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);
```

Méthodes constantes

Lorsqu'on utilise une référence constante, on ne peut accéder qu'aux méthodes 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
}
...
```

Propriétés d'un destructeur

La création d'un objet appelle un constructeur.

La supression d'un objet appelle un **destructeur**.

Définition et propriétés

Un destructeur est une méthode qui :

- n'a pas de type de retour,
- n'a pas d'argument,
- porte le nom de la classe précédé de `~` (tilde).

Implémentation des destructeurs

Propriétés des destructeurs

Un destructeur est :

- unique pour chaque classe,
- fourni par défaut par remplaçable,
- **JAMAIS** appelé explicitement.

```
class Obj{
    ...
public:
    Obj(); // constructeur vide
    Obj(int i);

    ~Obj(); // destructeur
    ...
};
```

```
Obj::~Obj(){

    cout << "Destruction";
    cout << endl;

}
```

Plan de la séance

Les chaînes de caractères

Les chaînes de caractères existent dans la bibliothèque standard `std::string`.

```
using namespace std;
string s = "toto"; // Création et affectation
char c = s[2]; // Récuparation du troisième caractère
int l = s.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.

Comparaisons des chaînes de caractères

1. Il est possible d'utiliser l'ordre lexicographique pour les comparer.

```
"a" < "b"   // --> TRUE
"d" > "a"   // --> FALSE
"a" < "ab"  // --> TRUE
"A" != "a"   // --> TRUE
"cat" < "caterpillar"  // --> TRUE
```

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

```
size_t i = s.find('h'); // i : indice de h dans s
size_t j = s.find('h',3); // j : indice de h dans s a partir de 3

size_t k = s.find("hop"); // k : indice de la sous-chaîne dans s
size_t l = s.find("hop", 3); // l : indice dans s à partir de 3
```

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

Chaînes de caractères

3. Concaténation

```
string a = "le début et";
string b = "la fin";

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

D'autres opérations (cf. polycopié)

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

Plan de la séance

Fichiers

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 f("chemin/fichier.txt");
ofstream f2;
f2.open("chemin/fichier.txt");

f << "ligne " << 1 << endl;
f << "ligne 2 ";
f << endl;

f.close();
```

Fichiers

Lecture dans un fichier

```
ifstream g("chemin/fichier");
int i;
double d;
g >> i >> d;
g.close();
```

Tester si le fichier est ouvert

```
ofstream f2;
f2.open("chemin/fichier.txt");
if(! g.is_open()){
    cout << "Erreur" << endl;
    return 1;
}
...
f.close();
```

TP

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.