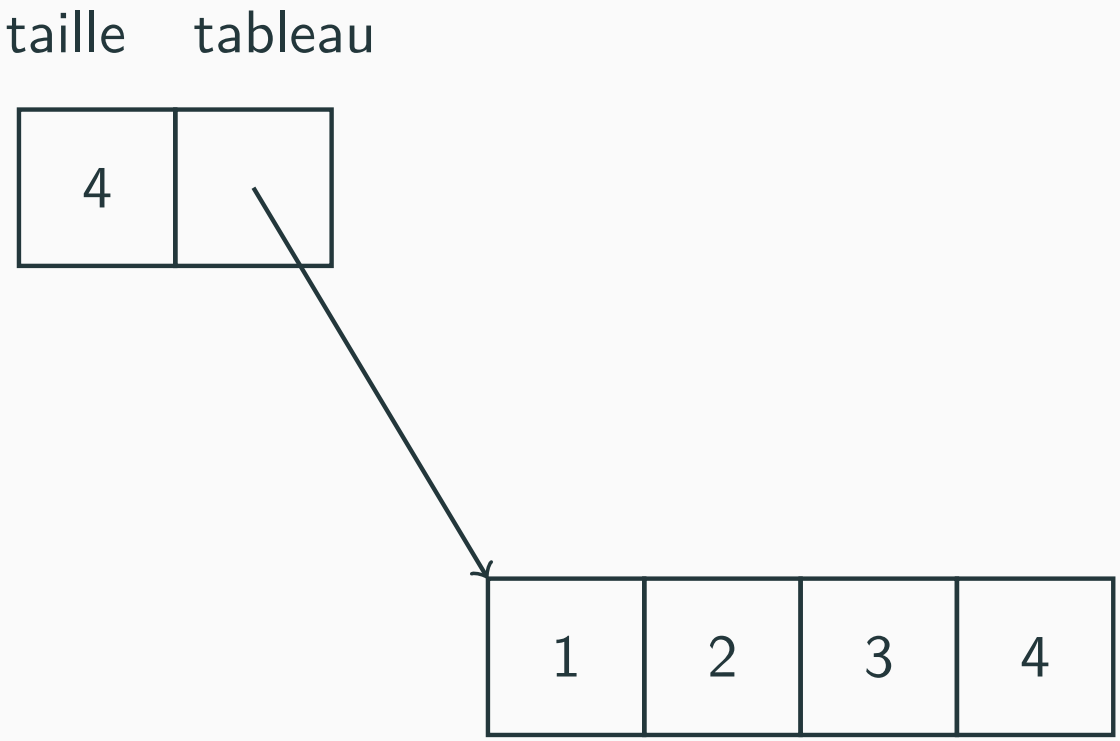
M1103 Amphi 2 : Classes

Thomas Nowak

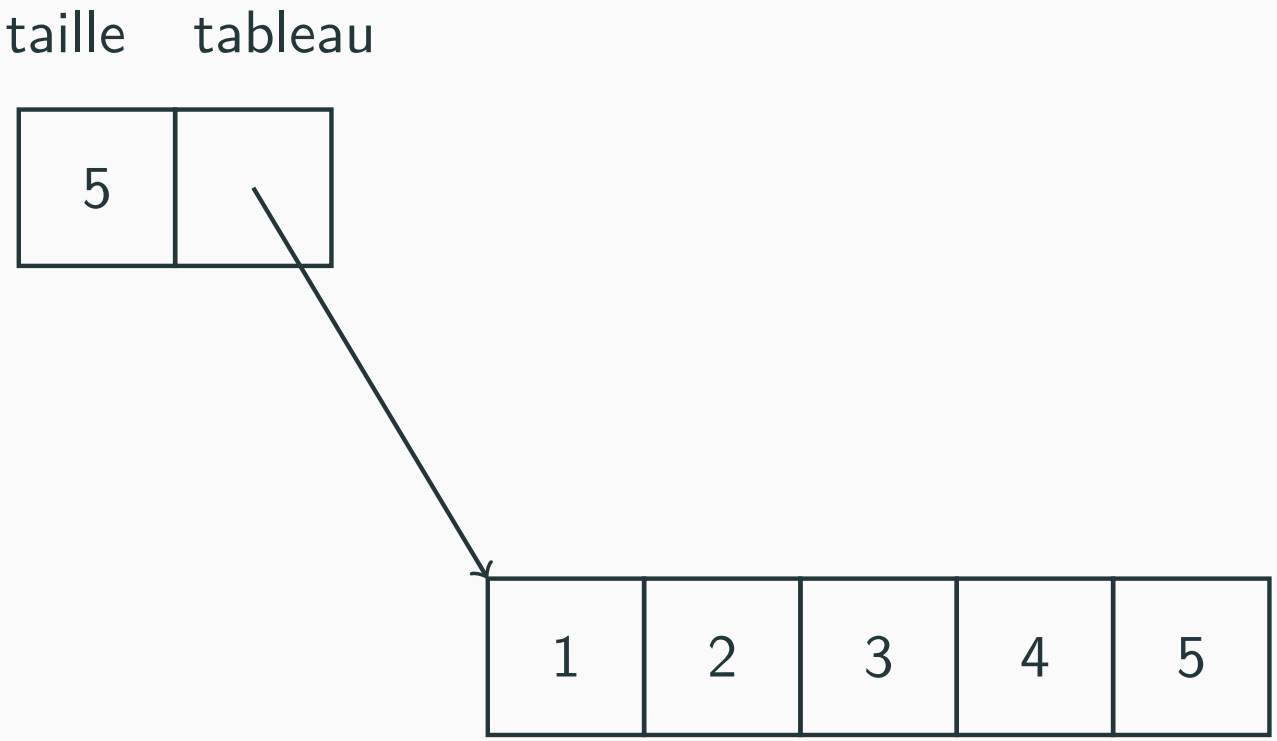
Université Paris-Sud

La dernière fois

Vecteurs



Vecteurs



Addendum Organisation

Addendum Organisation

1. Mise à jour Feuille 1
    - exercice “médiane” comme bonus
    - rajout d’un exercice plus élémentaire à sa place
2. Notes de cours
    - élaborer vos notes prises en amphi sous forme électronique, pour des points de bonus du CC
    - envoyez-les à thomas.nowak@lri.fr

Objets et classes

Objets et classes

- C++ connaît quelques types de base : `int`, `char`, `double`
- mais aussi quelques types plus compliqués : `string`, `vector`
- que fait-on si on veut travailler avec des structures de données qui ne sont pas déjà définies en C++ ?
- exemple : dates

Dates

```
int annee = 2018;
int mois = 11;
int jour = 14;
```

Dates

```
bool est_dimanche(int annee, int mois, int jour);
```

Dates

```
bool est_dimanche(Date d);
```

Classes

- comme le fait un vecteur qui *encapsule* un tableau brut et une taille, on peut *encapsuler* d'autres données dans une nouvelle structure
- définition d'une nouvelle structure = **classe**
- une classe définit
  - représentation : comment stocker les données
  - opérations : comment les utiliser

Définition d'une classe

```
class Date {
public:
    // accessible de l'exterieur
    int annee;
    int mois;
    int jour;

private:
    // inaccessible de l'exterieur
};
```

Déclaration et utilisation d'un objet

```
Date d;

d.annee = 2018;
d.mois = 11;
d.jour = 14;
```

Objets et fonctions

- valeur de retour d'une fonction :

```
Date f()
```

- passage par copie :

```
void f(Date d)
```

- passage par référence :

```
void f(Date &d)
```

- passage par référence constante :

```
void f(const Date &d)
```

Fonctions membres

Classes

- en mettant toutes les données publiques, on ne peut pas garantir la *validité* des données
  - parce qu'elles peuvent être changées de l'extérieur
- exemple :

```
d.mois = 11;
d.jour = 31;
```

- solution : mettre les données en *privé*
- question : comment peut-on utiliser les données maintenant ?

Classes

- réponse : définir des **fonctions membres**
- des fonctions qui agissent directement sur l'objet
- et ont accès aux données privées
- exemple :

```
bool dim = d.est_dimanche();
```

bien que

```
d.jour
```

soit inaccessible

Classes

- souvent :
  - définir des *getters* pour pouvoir accéder aux valeurs des données privées :

    ```
    int get_jour()
    {
        return jour;
    }
    ```

  - définition de classe + prototypes des fonctions membres → fichier `.h`
  - implémentation des fonctions membres → fichier `.cpp`

Implémentation de fonctions membres

- si implémentée dans le fichier `.cpp` (et déclarée dans `.h`) :
  - spécifier le nom de la classe à l'implémentation d'une fonction membre

```
class Date {
public:
    int get_jour();
};

int Date::get_jour() {
    return jour;
}
```

Constructeurs

Constructeurs

- comment initialiser les données d'un objet ?
- solution 1 : fonction membre `init` :

```
Date d;
d.init(2018, 11, 14);
```

- possible car fonction membre `init` peut accéder aux données privée de `Date`
- problème : pas de vérification si utilisation avant initialisation :

```
Date d;
cout << d.get_jour() << endl;
d.init(2018, 11, 14);
```

Constructeurs

- comment initialiser les données d'un objet ?
- solution 2 : *forcer* l'initialisation des données lors de la *création* de l'objet
- définition d'un **constructeur**

```
Date d(2018, 11, 14);
```

- plus de problème avec utilisation avant initialisation

Constructeurs

```
class Date {
private:
    int annee, mois, jour;
public:
    Date(int annee_, int mois_, int jour_);
};

Date::Date(int annee_, int mois_, int jour_) {
    annee = annee_;
    mois = mois_;
    jour = jour_;
}
```

Constructeurs

```
class Date {
private:
    int annee, mois, jour;
public:
    Date(int annee, int mois, int jour);
};

Date::Date(int annee, int mois, int jour) {
    this->annee = annee;
    this->mois = mois;
    this->jour = jour;
}
```

Fonctions membres constantes

Fonctions membres constantes

- la fonction

  ```
  void afficher(const Date &d)
  ```

  ne peut pas changer la date `d`

- et si on la transformait en fonction membre ?

  ```
  void Date::afficher()
  ```

- a priori cette fonction membre peut changer les données de la date `d`

Fonctions membres constantes

- il est possible d'interdire à cette fonction de changer les donnes de la date `d` :

  ```
  void Date::afficher() const
  ```

- cette déclaration `const` provoque une erreur de compilation si on veut modifier les données de l'objet courant
- pareil si on essaye de modifier un objet pris en référence constante

Questions