

Les templates

- Les templates permettent de paramétrer les structures et les fonctions.
- Objectif : rendre les fonctions et les structures indépendantes du ou des types de données manipulés.
- Un patron est utilisé par le compilateur pour l'instancier sur des types particuliers.

Les templates (2)

- Exemple : une fonction qui calcule le maximum sur un tableau.
- Pour l'utiliser sur plusieurs types, deux choix sont possibles :
 - surcharger la fonction pour chaque type nécessaire
 - créer un patron, le compilateur instanciera alors celui-ci pour chaque type employé dans le programme.
- Forme très puissante de généricité (méta-programmation).

Les templates (3)

- Pour créer un template, on préfix la déclaration de la structure ou de la fonction par
`template<class T1, ..., class Tn>`
- T1, ... Tn représentent les types qui seront employés dans les instantiations.
- Dans le reste de la fonction ou de la structure, on emploie T1, ... Tn comme n'importe quel type 'normal'.

Les templates (4)

- Pour instancier un template, on précise les types à employer après le nom de celui-ci.
- Exemples:
 - `Point<int> p;`
 - `afficher<double>(10.2);`
- Pour une fonction, on peut omettre la spécification des types quand cela est non-ambigu. Le compilateur va instancier les types adéquats.

Exemples sur les templates

```
// Sur une structure
template<class T> struct Complexe
{
    T re;
    T img;
};

Complexe<int> cpxEntier = {1, 2};
Complexe<double> cpxDouble = {1.0, 2.0};

// Sur une fonction
template<class T> T max(const T &a, const T &b)
{
    return (a < b) ? b : a;
}

int a = max<int>(3, 2);           // Type explicite
double b = max(10.0, 12.0);     // Type implicite
```

Templates et opérateurs

- Pour instancier un template de fonction sur un type donné, il faut que les opérateurs utilisés dans celui-ci existent pour le type à instancier.
- Sur l'exemple précédent, '<' doit être défini pour chaque type employé avec 'max' dans le programme.
- En définissant un template de structure, on peut définir des templates d'opérateurs sur celui-ci puisque ce sont des fonctions ...

Rendre un algorithme générique

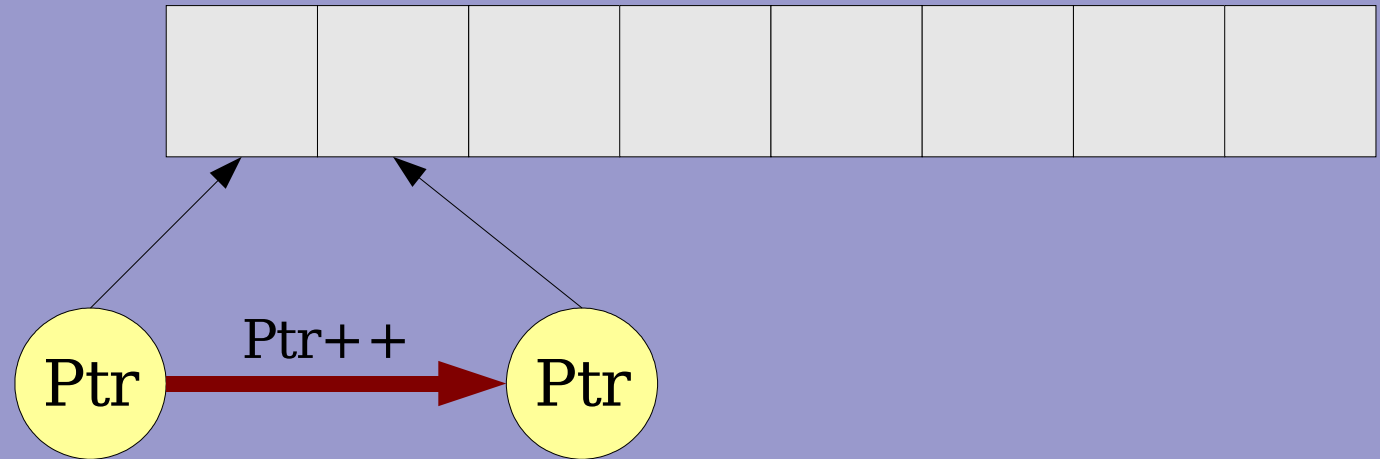
- Pour qu'un algorithme soit générique, il doit pouvoir opérer :
 - sur n'importe quel type de données
 - sur n'importe quelles structures de données.
- Pour les types de données : templates.
- Pour les structures de données : il faut utiliser une façon d'explorer les éléments indépendante de la façon dont ils sont organisés.

Rendre un algorithme générique (2)

- Les opérateurs guident l'exploration.
- Un tableau : exploration indicée des éléments (avec l'opérateur '[').
—> spécifique tableaux
- On peut aussi utiliser des pointeurs et s'affranchir des indices : '++' et '--' permettent de passer d'un élément à un autre. —> plus général
- Ainsi si pour une autre structure de données on peut se déplacer à l'aide de pointeurs et de '++' et '--' (éventuellement surchargés), alors l'algorithme peut être employé tel quel.

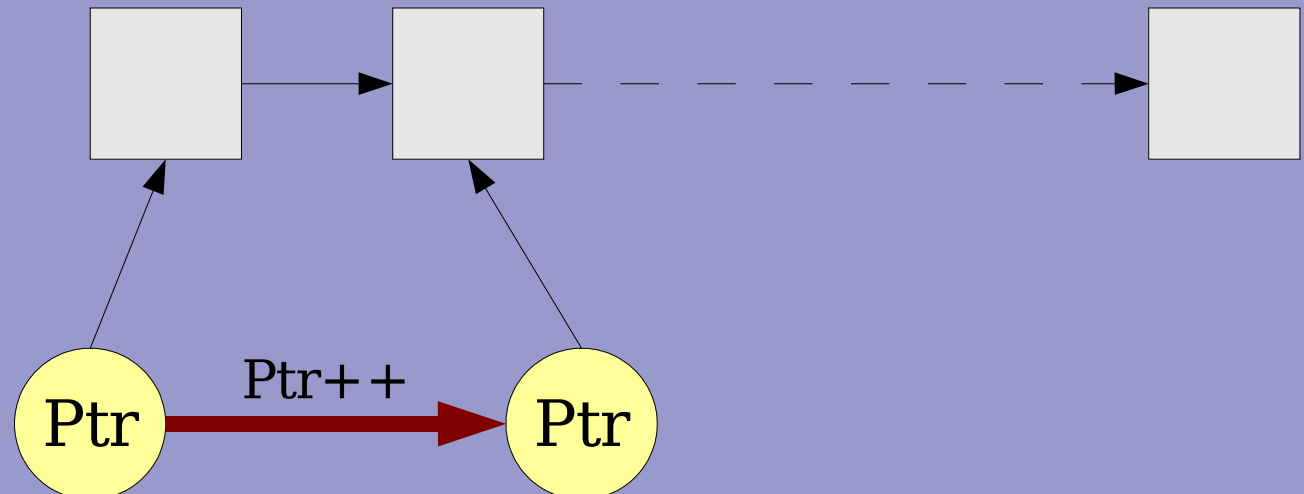
Exemple

Tableau :



Ici, l'algorithme utilise uniquement '+' sur un pointeur pour se déplacer dans la structure de données.

Liste :



Démarche pour la généricité

Pour rendre un algorithme générique, on emploie la démarche suivante.

- Développer et tester sur des types simples (int, ...).
- Rendre indépendant du type de données.
- Rendre indépendant de la structure de données employée.
- Faire un beau template.