Les classes et fonctions variadiques

http://en.cppreference.com/w/cpp/language/parameter_pack et http://en.cppreference.com/w/cpp/language/sizeof...

Fonctions avec nombre de paramètres non fixés

Permet d'écrire une fonction avec un nombre variable de paramètres de fonction. Exemple simple : on veut afficher un tableau avec des variables. Première approche, avec fonctions classiques et paramètre par défaut :

```
void print_array (int a, int b = 0, int c = 0, int d = 0) {
    std::cout << a << '\t' << b << '\t' << c << '\t' << d <<
'\n';
}


// ou avec surcharge de fonction
void print_array (int a) {
    std::cout << a << '\n';
}

void print_array (int a, int b) {
    std::cout << a << '\t' << b << '\n';
}

void print_array (int a, int b, int c) {
    std::cout << a << '\t' << b << '\t' << c << '\n';
}

void print_array (int a, int b, int c, int d) {
    std::cout << a << '\t' << b << '\t' << c << '\t' << d <<
'\n';
}
```

La première affiche toujours 4 colonnes (remplit avec des 0 si

nécessaire). La seconde évite les 0 inutiles, mais nécessite d'écrire une fonction pour chaque possibilité.

Améliorable pour ne pas prendre que des int :

```
template<class A>
void print_array (A a) {
    std::cout << a << '\n';
}

template<class A, class B>
void print_array (A a, B b) {
    std::cout << a << '\t' << b << '\n';
}

template<class A, class B, class C>
void print_array (A a, B b, C c) {
    std::cout << a << '\t' << b << '\t' << c << '\n';
}

template<class A, class B, class C, class D>
void print_array (A a, B b, C c, D d) {
    std::cout << a << '\t' << b << '\t' << c << '\t' << d <<
'\n';
}
```

Très lourd à écrire, nombre maximal de paramètres fixés

Les fonctions variadiques

Possibilité d'indiquer qu'une fonction prend un nombre indéterminé de paramètres. A la compilation, le compilateur déterminer lui même le nombre de paramètre.

Comme on ne connaît pas le nombre de paramètres, le plus simple est d'utiliser une fonction récursive, avec une condition d'arrêt. L'idée est que l'on définie une fonction qui prend N paramètres et utilise la fonction N-1 paramètres (fonction récursive) et une fonction qui prend 1 paramètre et s'appelle pas de fonction N (condition d'arrêt).

En pseudo code :

```
f(N paramètres) {
    appel de f(N-1 paramètres)
}

f(1 paramètre) {
}
```

Si on appelle ce code avec 5 paramètres par exemple, le code précédent est “déroulé” (ie convertie en une suite de fonctions non récursives) de la façon suivante :

```
f(5 paramètres) {
    appel de f(4 paramètres)
}

f(4 paramètres) {
    appel de f(3 paramètres)
}

f(3 paramètres) {
    appel de f(2 paramètres)
}

f(2 paramètres) {
    appel de f(1 paramètres)
}

f(1 paramètre) {
}
```

On voit que chaque fonction appelle la fonction N-1 sauf pour N=2, qui appelle la fonction déjà définie f1.

Syntaxe

Utilisation de `…` dans la liste des paramètres template et la liste des paramètres de fonction.

```
template<class ... T>
void f(T ... values) {
}
```

Appel récursif : extraire le premier paramètre et les autres paramètres, appel récursif sur les autres paramètres.

```
template<class T>
void f(T value) {                    // condition d'arrêt
}

template<class T, class ...Args>
void f(T value, Args ... args) {
    f(args);                         // appel récursif
}
```

Avec le code de print_array :

```
#include <iostream>

template<class T>
void print_array (T value) {
    std::cout << value << '\n';   // affichage du dernier
élément
}

template<class T, class ...Args>
void print_array (T value, Args ... args) {
    std::cout << value << '\t';   // affichage de l'élément
courant
    print_array(args...);            // affichage des autres
éléments
}

int main() {
    print_array(1, 2, 3);
    print_array(1.2, "hello", 2.3, "world");
}
```

Template spécialisation ?

Classes template variadiques

Idem, avec des classes.

```
template<class ... Args>
struct A {
    std::tuple<Args...> args;
};
```

Exercices

- réécrire std::tuple
- écrire make_tuple

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++