- **Les patrons de fonctions**
- **Les classes paramétrées**

LES PATRONS DE FONCTIONS

(fonction générique ou fonction modèle ou fonction template)

Lorsque l'algorithme est le même pour plusieurs types de données, il est possible de créer un patron de fonction.

C'est un modèle à partir duquel le compilateur générera les fonctions qui lui seront nécessaires.

Exemple 1 :

```
template <class T>
void affiche(T *tab, unsigned int nbre) {
  for(int i = 0; i < nbre; i++)
    cout << tab[i] << " ";
  cout << endl;
}

void main() {
  int tabi[6] = {25, 4, 52, 18, 6, 55};
  affiche(tabi, 6);

  double tabd[3] = {12.3, 23.4, 34.5};
  affiche(tabd, 3);

  char *tabs[] = {"Bjarne", "Stroustrup"};
  affiche(tabs, 2);
}
```

Exemple 2 :

```
template <class T>     // déclaration du patron
T min(T, T);

// ...    la suite de votre programme

template <class T>     // définition du patron
T min(T t1, T t2) {
  return t1 < t2 ? t1 : t2;
```

```
}
```

Exemple 3 :

```
template <class X, class Y>
int valeur(X x, Y y)
{  // ... }
```

Les fonctions génériques peuvent, tout comme les fonctions normales, être surchargées.

Ainsi il est possible de spécialiser une fonction à une situation particulière, comme dans l'exemple qui suit :

```
template <class T>
T min(T t1, T t2) {
  return t1 < t2 ? t1 : t2;
}

char *min(char *s1, char *s2) {
  // fonction spécialisée pour les chaînes
  // il est évident la comparaison des adresses
  // des chaînes est absurde ...
  return strcmp(s1, s2)<0 ? s1 : s2;
}

void main() {
  cout << min(10, 20) << "  " << min("Alain", "Bjarne") << endl;
}
```

LES CLASSES PARAMETREES

Les classes paramétrées permettent de créer des classes générales et de transmettre des types comme paramètres à ces classes pour construire une classe spécifique.

Définition d'un modèle de classe

Le mot clé *template* suivi des paramètres du type du modèle débute la définition de la classe modèle.

Exemple :

La classe *List* n'a pas à se soucier du type réel de ses éléments.

Elle ne doit s'occuper que d'ajouter, supprimer, trier ses éléments.

On a donc intérêt à paramétrer la classe *List* par le type des éléments qu'elle stocke.

```
template <class T> class List {
  public:
    List();              // constructeur
    List(const List &l); // constructeur de copie
    ~List();

    void add(T elmt);    // ajoute un élément à la liste
    T &get();            // retourne l'élément courant
    // ...
  private:
    // ...
};
```

La **définition des méthodes** se fait de la façon suivante :

```
template <class T> void List<T>::add(T elemt) {
  /*...*/
}

template <class T> T &List<T>::get() {
  /*...*/
}
```

Une **utilisation de cette classe** sera, par exemple :

```
void main() {
  List<int> resultat;

  List<Employe> repertoire;
```
