Définition des fonctions

La **définition** d'une fonction sert, comme son nom l'indique, à définir ce que fait la fonction :

- spécification du **corps** de la fonction

Syntaxe :

```
type nom ( liste de paramètres )
{
    instructions du corps de la fonction;
    return expression;
}
```

Exemple :

```
double moyenne (double x, double y)
{
   return (x + y) / 2.0;
}
```

Corps de fonction

Le corps de la fonction est donc un **bloc**
dans lequel on peut utiliser les paramètres de la fonction (en plus des variables qui lui sont propres).

La valeur retournée par la fonction est indiquée par l'instruction :

```
return expression;
```

où l'*expression* a le même *type* que celui retourné par la fonction.

```
double moyenne (double x, double y)
{
   return (x + y) / 2.0;
}
```

L'instruction `return` fait deux choses :

- elle précise la valeur qui sera fournie par la fonction en résultat
- elle met fin à l'exécution des instructions de la fonction.

L'expression après `return` est parfois réduite à une seule variable ou même à une valeur littérale, mais ce n'est pas une nécessité.

Remarques sur l'instruction `return` (1/4)

Il est possible de placer *plusieurs* instructions `return` dans une même fonction.

Par exemple, une fonction déterminant le maximum de deux valeurs peut s'écrire avec une instruction `return` :

```
double max2(double a, double b)
{
  double m;
  if (a > b) {
    m = a;
  } else {
    m = b;
  }
  return m;
}
```

ou deux :

```
double max2(double a, double b)
{
  if (a > b) {
    return a;
  } else {
    return b;
  }
}
```

Remarques sur l'instruction `return` (2/4)

Le type de la valeur retournée doit correspondre au type dans l'en-tête :

```
double f() {
  bool b(true);
  ...
  return b; // Erreur : mauvais type
}
```

Remarques sur l'instruction `return` (3/4)

`return` doit être la toute dernière instruction exécutée :

```
double lire() {
  cout << "Entrez un nombre : ";
  double n(0.0);
  cin >> n;
  return n;
  cout << "entré : " << n << endl; // Jamais exécuté
}
```

Remarques sur l'instruction `return` (4/4)

Le compilateur doit être sûr de toujours pouvoir exécuter un `return` :

```
double lire() {
  cout << "Entrez un nombre : ";
  double n(0.0);
  cin >> n;
  if (n > 0.0) {
    return n;
  }
  // Erreur : pas de return si n <= 0 !
}
```

Fonctions sans valeur de retour

Quand une fonction ne doit fournir aucun résultat (on appelle de telles fonctions des « **procédures** ») :

☞ définir une fonction **sans valeur de retour**

On utilise alors le type particulier `void` comme type de retour.

Dans ce cas la commande de retour `return` est optionnelle :

- soit on ne place aucun `return` dans le corps de la fonction
- soit on utilise l'instruction `return` sans la faire suivre d'une expression : `return;`

Exemple :

```
void affiche_racine(double a)
{
  if (a < 0.0) {
    return; /* Grâce à ce return, on quitte la fonction avant *
             * de calculer sqrt(a) si a est négatif.          */
  }
  cout << sqrt(a);
  // il n'est pas nécessaire de mettre un return ici
}
```

Fonctions sans paramètre

Il est aussi possible de définir des fonctions **sans paramètre**.
Il suffit, dans le prototype et la définition, d'utiliser une liste de paramètres vide : `()`

Exemple :

```
double saisie()
{
  double nb_points(0.0);

  do {
    cout << "Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  return nb_points;
}
```

La fonction `main()`

`main` est aussi une fonction avec un nom et un prototype imposés.

Par convention, tout programme C++ doit avoir une fonction `main`, qui est appelée automatiquement quand on exécute le programme.

Cette fonction doit retourner une valeur de type `int`. La valeur 0 indique par convention que le programme s'est bien déroulé.

Les deux seuls prototypes autorisés pour `main` sont :

```
int main();
int main(int argc, char** argv);
```

Seul le premier sera utilisé dans ce cours.