IFT159
Analyse et programmation
Chapitre 3 — Fonctions en C++

Gabriel Girard

Département d'informatique

UNIVERSITÉ DE SHERBROOKE

7 septembre 2008

Chapitre 3 — — Fonctions en C++

Abstraction procédurale

- Les fonctions permettent de faire de l'**abstraction procédurale**
- L'abstraction procédurale consiste à associer un nom de fonction à un algorithme complexe et à l'utiliser sans se soucier des détails d'implémentation.
- Avec l'abstraction procédurale, un programme est écrit comme une suite d'appels à des fonctions indépendantes.
- Avantages de l'abstraction procédurale
  - Réutilisation
  - Masquage d'information (lisibilité)

Utilité des fonctions

- Faire abstraction de certaines étapes moins importantes.
- Décomposition
- Clarté, lisibilité
- Les principes des modules (en conception) s'appliquent aux fonctions.

Correspondance avec les mathématiques

- Une fonction informatique est plus général que la fonction mathématique.
- Si la fonction n'a pour entrée que des paramètres et pour sortie qu'une valeur de retour, la fonction mathématique est un bon modèle.
- S'il existe des interactions avec l'environnement (entrée ou sortie), la fonction mathématique n'est plus un modèle correct.
- Autres cas absurdes en math : une fonction sans paramètre ou une fonction sans retour.

Concept de fonctions

Le concept de fonction permet :

- De regrouper des énoncés C++ dans une seule unité.
- D'implémenter une solution avec une approche modulaire et descendante (programmation modulaire).

Fonctions sans paramètre — Syntaxe

Déclaration (ligne prototype)

type nom_fonction() ;

Exemple

- `void dessine_maison();`
- `int mois();`
- `float test2();`

Fonctions sans paramètre — Syntaxe

Appel de la fonction

```
nom_fonction();
```

Exemple

- `dessine_maison();`
- `date = mois() * nb_jour;`
- `resultat = test2() * 3.3 / test2();`

Fonctions sans paramètre — Syntaxe

Définition de fonction

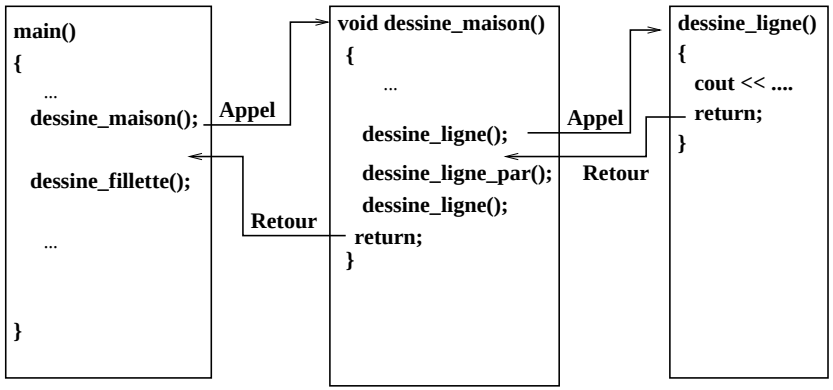
```
type nom_fonction()                  // entête
{
     déclarations de fonctions       // corps
     déclarations de constantes      // corps
     déclarations de variables       // corps
     énoncés exécutables             // corps
     return valeur ;                 // corps
}
```


Fonctions sans paramètre — Conventions

Normes à respecter pour le cours :

- Noms significatifs ;
- Dans la définition, on place des commentaires :
  - Au début (ce qu'elle fait, E/S, auteur) ;
  - À la fin (optionnellement) (ex. : // fin nom_fonction) ;
- Déclaration dans la fonction qui l'utilise ;
- Un commentaire accompagne la déclaration.

Séquence d'exécution



Fonctions avec paramètres

- Le concept de paramètres permet de passer de l'information à une fonction.
- Paramètres : proches des variables d'une fonction mathématique.

Fonctions avec paramètres
Syntaxe

Déclaration (ligne prototype)

type nom_fonction(types_des_paramètres) ;

Exemple

- `void uneFonction(int, float, char, string);`
- `int mois(int);`
- `float test2(float, float, float);`

Fonctions avec paramètres
Syntaxe

Appel de fonction

nom_fonction(paramètres_effectifs) ;

Exemple

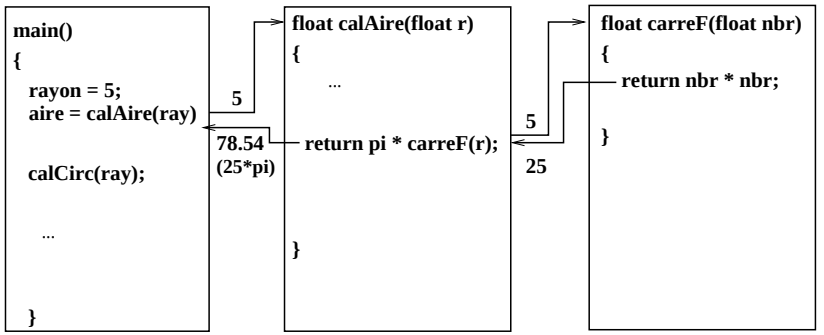
```
◇ uneFonction(p1, p2, p3, p4);
◇ date = mois(244) * nb_jour;
◇ resultat = test2(2.2, p2, 10.0) * 3.3 /
                              test2(1.2, 3, 4.4);
```


Fonctions avec paramètres
Syntaxe

Définition de fonction

```
type nom_fct(paramètres formels)    // entête
{
   déclarations de fonctions        // corps
   déclarations de constantes       // corps
   déclarations de variables        // corps
   énoncé exécutables               // corps
   return valeur ;                  // corps
}
```


Séquence d'exécution



Portée des variables et constantes

- Les identificateurs déclarés à l'intérieur d'une fonction sont appelés des identificateurs locaux et ils n'existent que pour cette fonction.
- Les paramètres formels n'existent qu'à l'intérieur de la fonction.
- Les identificateurs déclarés à l'extérieur des fonctions sont appelés des identificateurs globaux et ils existent pour toutes les fonctions.
  - On peut utiliser des constantes globales.
  - Cela évite des répétitions inutiles.

Typage

- C++ est un langage fortement typé.
  - $\Rightarrow$ Il associe un type à chaque variable ou constante.
  - $\Rightarrow$ La variable doit être utilisée d'une façon consistante à son type.
  - $\Rightarrow$ Il se produira des erreurs s'il y a des incohérences.
- Changement explicite de type $\rightarrow$ *type (nomVariable)*.
- On peut aussi retrouver $\rightarrow$ *(type) nomVariable*.
- On appelle ceci une conversion explicite ou un *casting*.

Exemple de correspondance math-info

La fonction mathématique

$$\begin{array}{rccc} f: & \mathbb{N} & \rightarrow & \mathbb{R} \\ & x & \mapsto & x \times 2.33 \end{array}$$

se traduit par la fonction informatique

```
float f ( int x )
{
    return x * 2.33;
}
```


Exemple de correspondance math-info

La fonction mathématique

$$\begin{array}{rccc} f: & \mathbb{N} \times \mathbb{R} & \rightarrow & \mathbb{R} \\ & (x, y) & \mapsto & xy + 1 \end{array}$$

se traduit par la fonction informatique

```
float f ( int x,  float y )
{
    return x*y+1;
}
```

Exemple de correspondance math-info

La fonction mathématique

$$\begin{array}{rccl} \lfloor . \rfloor : & \mathbb{R} & \rightarrow & \mathbb{N} \\ & x & \mapsto & n \in \mathbb{N} \text{ tel que } n \leq x < n+1 \end{array}$$

se traduit par la fonction informatique

```
int partie_entiere ( float y )
{
    return (int) floor(y) ;
}
```


Module « ratio par tête »

Entrée :
- (*paramètre*) N_p, Nbre de personnes (*entier*)
- (*paramètre*) C, Charge du serveur (*réel*)

Sortie : (*valeur de retour*) R, le ratio par tête (*réel*)

Formules :
1. $R = \frac{T}{N_p}$
2. la tension $T = C^2 sinus(C)$

Module « ratio par tête »

- Algorithme :
  1. Calculer le carré de la charge du serveur.
  2. Calculer le sinus de la charge (Module « Calculer sinus »).
  3. Multiplier les deux précédents résultats (formule 2).
  4. Le ratio est le résultat précédent divisé par le nombre de personnes (formule 1).

Fonction implémentant « ratio par tête »

Voici la fonction en C++

```
float ratio_par_tete (int nbre_personnes, float charge)
{
    // declaration de la fonction appelee
    float calculer_sinus (float) ;

    // declaration des variables
    float ratio ;   // sortie

    float tension ;
    float carre_charge ;
    float sinus_charge ;
```

Fonction implémentant « ratio par tête »

```
    // Debut du cours de la fonction
    // 1. Calculer le carré de la charge du serveur.
    carre_charge = charge * charge;

    // 2. Calculer le sinus de la charge.
    sinus_charge = calculer_sinus(charge);

    // 3. Multiplier les deux précédents résultats.
    tension = carre_charge * sinus_charge;

    // 4. Le ratio est le résultat précédent divisé
    //    par le nombre de personnes.
    ratio = tension / nbre_personnes;

    return ratio;
}
```


Phases de développement

1. Comprendre le problème
2. Faire analyse du problème (entrées, sorties, formules, ...)
3. Décomposition du problème en sous-problèmes. On identifie les modules.
   1. Conception du problème principal (algorithme exprimé en termes d'appels à des modules).
   2. Pour chaque sous-problème (module) on refait la conception.
4. On code séparément chacun des modules grâce aux fonctions.