Chapitre 7 : Fonctions

Informatique de base
2013-2014

Sup Galilée

Qu'est-ce qu'une fonction ?

Tout langage de programmation offre le moyen de découper un programme en unités indépendantes qui peuvent s'appeler les unes les autres. En C, ces unités sont les fonctions.

Une fonction mathématique $f : E \to F$ associe à chaque élément $x \in D_f \subseteq E$ (D_f est le domaine de définition de f) un élément $y \in F$, que l'on note $f(x)$, tel que si $x = x' \in D_f$, alors $f(x) = f(x')$.On note parfois une telle correspondance fonctionnelle par $x \mapsto f(x)$, et le symbole x dans cette notation est appelé argument formel de la fonction f, alors qu'un élément $x \in D_f$ est appelé argument effectif, et en informatique on dit alors que $f(x)$ est un appel de fonction.

Les fonctions du langage C sont une généralisation des fonctions mathématiques puisqu'on autorise aussi des fonctions sans argument ou des fonctions sans résultat.

En C :

- Les ensembles mis en correspondance par les fonctions sont les ensembles de valeurs désignés par chaque type.
- Les variables d'une fonction sont appelées arguments formels et les valeurs (de ces variables) que l'on applique à cette fonction sont appelées arguments effectifs.
- Le corps d'une fonction est un bloc d'instructions, paramétré par les noms des arguments formels de cette fonction, dont l'exécution peut retourner (ou renvoyer) la valeur de l'application de cette fonction à ses arguments effectifs.
- L'application d'une fonction f d'arguments formels $x_1, \cdots, x_n$ est notée par l'expression $f(a_1, \cdots, a_n)$, dite appel de fonction, dans laquelle $a_1, \cdots, a_n$ sont des expressions dont les valeurs constituent les arguments effectifs de cette application. Dans cette notation il est nécessaire que l'expression a_i soit du même type que celui de l'argument formel x_i.
- L'évaluation d'un appel de fonction consiste à exécuter le corps de cette fonction dans un environnement dans lequel les arguments effectifs sont affectés aux arguments formels.

Un exemple : la fonction minimum

La fonction `minimum` peut être définie en C de la façon suivante :

```
(1)  int minimum (int x1, int x2)
(2)    {
(3)    if (x1 < x2)
(4)      return x1;
(5)    else
(6)      return x2;
(7)    }
```

Un exemple : la fonction minimum

```
(1)  int minimum (int x1, int x2)
(2)    {
(3)    if (x1 < x2)
(4)      return x1;
(5)    else
(6)      return x2;
(7)    }
```

Un exemple : la fonction minimum

La ligne (1) déclare le type d'arrivée (ici `int`), le nom de la fonction (ici `minimum`), le type et le nom des arguments formels (ici les arguments formels x1 et x2 sont de type `int`) de cette fonction.

Un exemple : la fonction minimum

```
(1)  int minimum (int x1, int x2)
(2)    {
(3)    if (x1 < x2)
(4)      return x1;
(5)    else
(6)      return x2;
(7)    }
```

Un exemple : la fonction minimum

Les lignes (2) à (7) constituent le corps de la fonction.

L'évaluation de l'appel minimum (19,5) consiste à exécuter le corps de la fonction dans l'environnement où `x1=19` et `x2=5`, ce qui déclenche l'exécution de la fonction `return 5`. La valeur de cet appel de fonction est donc 5 (c'est la valeur retournée par la fonction).

Un programme C est constitué d'une suite de définitions de variables globales ou de fonctions.

Ces définitions établissent l'environnement global dans lequel s'exécute ce programme.

Les variables globales et les fonctions sont définies en dehors du corps d'une fonction (y compris `main`) et peuvent être utilisées dans le corps des fonctions définies dans le programme (dont la définition apparaît après celles de ces variables ou fonctions).

Définition d'une fonction

La définition d'une fonction suit la syntaxe ci-dessous

```
T nom-fonction (decl-1,..., decl-n)
  corps-de-la-fonction
```

où

- dans la première ligne, appelée en-tête de la fonction, `T` est un nom de type (c'est le type d'arrivée de la fonction), et `decl-1`, ..., `decl-n` sont les déclarations des arguments formels,
- corps-de-la-fonction est un bloc d'instructions (le corps de la fonction), dont l'exécution calcule la valeur de la fonction (qui doit être de type `T`) et éventuellement ses effets de bord.
- Si le type `T` est différent du type vide void, alors on doit trouver au moins une instruction `return exp;` dans le corps de la fonction (cette instruction permettant de retourner le résultat de la fonction) où exp est une expression de type `T`.

Un autre exemple

La fonction qui calcule la moyenne de deux entiers peut être définie de la manière suivante :

```
float moyenne (int x1,int x2)
  {
  float m;
  m=(x1+x2)/2.0;
  return m;
  }
```

Cette fonction peut également être définie simplement par :

```
float moyenne (int x1,int x2)
  {
  return (x1+x2)/2.0;
  }
```

Rappelons que la définition d'une fonction `f` ajoute la ligne suivante à l'environnement global :

```
fonction    f
```

Cas particuliers : fonctions sans valeur de retour

Une fonction peut ne pas posséder de valeur de retour. Son en-tête prend alors la forme suivante :

```
void nom-fonction (decl-1,...,decl-n)
```

void est le type vide. Dans ce cas, il ne faut pas d'instruction `return` puisque cette fonction ne retourne pas de valeur.

Par exemple, une fonction qui se contente d'afficher un message :

```
void ecrire_entier (int n)
  {
  printf("L'entier en argument est %d\n",n);
  }
```

Cas particuliers : fonctions sans argument

Une fonction peut ne pas posséder d'argument. Son en-tête prend alors la forme suivante :

```
T nom-fonction (void)
```

Par exemple, la fonction suivante lit un entier sur l'unité d'entrée standard (en général le clavier) :

```
int lire_entier (void)
  {
  int n;
  scanf("%d", &n);
  return n;
  }
```

Les deux fonctions précédentes peuvent être combinées pour afficher l'entier lu : `ecrire_entier(lire_entier());`

On peut également définir une fonction sans argument ni sans valeur de retour. La fonction `main` a pour type d'arrivée `int` et peut avoir des arguments. Lorsque cette fonction n'a pas d'argument et ne renvoie pas de valeur il est usuel d'écrire son en-tête `main ()`

Appel d'une fonction

L'appel d'une fonction est une expression de la forme

`nom-fonction (exp-1,...,exp-n);` où

`nom-fonction` désigne le nom d'une fonction, et `exp-1,..., exp-n` sont des expressions dont les valeurs constituent les arguments effectifs de la fonction. L'expression `exp-i` doit être du même type que le ième argument formel déclaré par `decl-i` dans l'en-tête de la fonction `nom-fonction`.

Appel d'une fonction

Si `f` est le nom d'une fonction dont les arguments formels sont les variables $x_1, \ldots, x_n$, de type T_1, ..., T_n, et dont le corps est le bloc `b`, alors la valeur de l'expression `f(exp-1,...,exp-n)` est calculée de la façon suivante :

1. Les valeurs des arguments effectifs sont calculées : v_1=val(exp-1), ..., v_n=val(exp-n).
2. Le bloc

   T_1 x_1;
   ...
   T_n x_n;
   x_1=v_1;
   ...
   x_n=v_n;
   b

   est exécuté.
3. La valeur retournée est celle produite par la première instruction `return exp;` exécutée. On a donc au final : val(f(exp-1,...,exp-n))=val(exp).

Appel d'une fonction

Les appels de fonction produisent une pile d'environnements. Regardons cela sur un exemple :

```
const float pi=3.14;
float aire (floar r)
  {
  return pi * r * r;
  }
float volume_cylindre (float r, float h)
  {
  float v;
  v = h * aire (r);
  return v;
  }
main ()
  {
  float v;
  volume (3.0,10.0);
  }
```

Appel d'une fonction

Ce programme est constitué d'une variable (constante !) globale `pi` et de trois fonctions : la fonction `aire` qui calcule l'aire d'un disque de rayon `r`, la fonction `volume_cylindre` qui calcule le volume d'un cylindre de rayon `r` et de hauteur `h` à l'aide de la fonction `aire`, et la fonction `main` qui calcule le volume d'un cylindre de rayon 3 et de hauteur 10 à l'aide de la fonction `volume_cylindre`.

Appel d'une fonction

L'évolution de l'état de ce programme au cours de son exécution est :

Immédiatement après l'appel `aire(3.0);`

variable	r	3.0

---

variable	v	(–)
variable	h	(10.0)
variable	r	(3.0)

---

variable	v	(–)

---

fonction	main()
fonction	volume_cylindre()
fonction	aire()
constante	pi 3.14

Appel d'une fonction

Immédiatement après l'exécution de `return pi * r * r;` dans `aire`

		28.26
variable	v	–
variable	h	10.0
variable	r	3.0

---

variable	v	(–)

---

fonction	main()	
fonction	volume_cylindre()	
fonction	aire()	
constante	pi	3.14

Appel d'une fonction

Immédiatement après l'exécution de `v = h * aire (r);` dans `volume_cylindre`

variable	v	282.6
variable	h	10.0
variable	r	3.0

---

variable	v	(–)

---

fonction	main()	
fonction	volume_cylindre()	
fonction	aire()	
constante	pi	3.14

Appel d'une fonction

Immédiatement après l'exécution de `v = volume (3.0,10.0);` dans `main`

variable	v	282.6

---

fonction	main()	
fonction	volume_cylindre()	
fonction	aire()	
constante	pi	3.14