Éléments d'informatique – Cours 11. Fonctions récursives.

Pierre Boudes

30 novembre 2009

Rappels sur les fonctions en C (1)

En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. (wikipédia)

Intérêt des fonctions :

- *factorisation*
- *réutilisation*
- *lisibilité*
- *structuration*

Rappels sur les fonctions en C (2)

Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs)

Rappels sur les fonctions en C (2)

Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs)

Convention de nommage

Il est pratique à l'usage de faire commencer le nom de chaque fonction par un verbe à l'infinitif. `afficher_...`, `lire_...`, etc. Exceptions : la fonction est connue sous un autre nom (`racine(x)`, `moyenne(...)`) ou se présente comme une fonction constante (sans paramètres)

Procédures

Les fonctions peuvent ne pas avoir de valeur de retour, on parle alors plutôt de procédures. D'un point de vue mathématique, ça n'a pas de sens.
Le résultat de l'exécution de la fonction est communiqué par effet de bord (typiquement une écriture à l'écran).

```
void afficher_resultat(int n)
{
    printf("Resultat : %d\n", n);
}
```

Fonctions sans paramètres

Il est également possible de définir des fonctions sans paramètres. D'un point de vue mathématique ce seraient des constantes. Soit réellement constantes (dans ce cas il vaut mieux utiliser des constantes symboliques) soit faisant des effets de bord (entrée clavier). Ces fonctions souvent sont aussi des procédures.

Fonctions sans paramètres

Il est également possible de définir des fonctions sans paramètres. D'un point de vue mathématique ce seraient des constantes. Soit réellement constantes (dans ce cas il vaut mieux utiliser des constantes symboliques) soit faisant des effets de bord (entrée clavier). Ces fonctions souvent sont aussi des procédures.

```
#define PI 3.14 /* plutot que: double pi() {return 3.14;} */
...
int lire_choix() /* effet de bord */
{
    int n;
    scanf("%d", &n);
    return n;
}
...
void afficher_menu() /* procedure */
{
    ...
}
```

Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

Terminaison

Il faut un cas de base qui ne déclenche pas d'appel récursif.

Fonctions récursives

Définition

Une fonction récursive est une fonction dont la définition fait appel à la fonction *elle-même*.

Il y a une forte analogie avec les maths : $(n+1)! = (n+1) \times n!$

Terminaison

Il faut un cas de base qui ne déclenche pas d'appel récursif.

Comme dans :

$$f : \mathbb{N} \to \mathbb{N}$$

$$n \mapsto \begin{cases} n \times f(n-1) \text{ si n} > 0 \\ 1 \text{ sinon} \end{cases}$$

Factorielle (1)

```
int factorielle(int n)
{
    int res; /* resultat */
    if (n > 0) /* cas recursif */
    {
        res = n * factorielle(n - 1);
    }
    else /* cas de base */
    {
        res = 1;
    }
    return res;
}
```

Factorielle (2)

Version plus concise :

```
int factorielle(int n)
{
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

Récursion (2). Pour aller plus loin

- Outre les exemples mathématiques directs comme factorielle, de nombreux problèmes sont beaucoup plus facile à résoudre de manière récursive. À au moins un moment du raisonnement, on suppose que l'on dispose déjà de la fonction qui résout le problème et on l'applique à un cas « plus petit ».
- On apprend ici la programmation impérative où un élément central est le changement d'état des cases mémoires (et les effets de bord comme vous le verrez au second semestre). En *programmation fonctionnelle*, l'accent est mis sur les fonctions sans effets de bord, et la récursion occupe le tout premier plan, notamment pour faire ce que l'on a l'habitude de faire avec des boucles en programmation impérative.
- Un appel récursif peut être indirect, c'est à dire effectué dans le code d'une fonction auxilliaire (hors programme).

Exemples. Affichage à la descente

Avec le code :

```
int factorielle(int n)
{
    printf("%d ", n);
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

Exemples. Affichage à la descente

Avec le code :

```
int factorielle(int n)
{
    printf("%d ", n);
    if (n < 2) /* cas de base */
    {
        return 1;
    }
    return n * factorielle(n - 1);
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

```
5 4 3 2 1
```

Exemples. Affichage à la remontée

Et avec le code :

```
int factorielle(int n)
{
    int res = 1;
    if (n > 1) /* cas recursif */
    {
        res =  n * factorielle(n - 1);
    }
    printf("%d ", n);
    return res;
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

Exemples. Affichage à la remontée

Et avec le code :

```
int factorielle(int n)
{
    int res = 1;
    if (n > 1) /* cas recursif */
    {
        res =  n * factorielle(n - 1);
    }
    printf("%d ", n);
    return res;
}
```

L'appel `factorielle(5)` aura pour effet de bord d'afficher :

`1 2 3 4 5`

- Comment obtenir `5 4 3 2 1 1 2 3 4 5` ?
- Peut-on obtenir : `1 2 3 4 5 5 4 3 2 1` ?

Exemples. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Exemples. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Relation de récurrence donnée par le triangle de Pascal :

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

Exemples. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Relation de récurrence donnée par le triangle de Pascal :

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

Cas de base : $\binom{n}{0} = \binom{n}{n} = 1$

Exemples. Double appel

Coefficients binomiaux :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Relation de récurrence donnée par le triangle de Pascal :

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

Cas de base : $\binom{n}{0} = \binom{n}{n} = 1$

Code :

```
int binomial(int n, int p)
{
    if ( (p == 0) || (n == p) ) /* cas de base */
    {
        return 1;
    }
    return binomial(n - 1, p - 1) + binomial(n - 1, p);
}
```

Exemple. Écriture récursive de boucles

Calcul de la moyenne d'une série saisie par l'utilisateur.

Exemple. Écriture récursive de boucles

Calcul de la moyenne d'une série saisie par l'utilisateur.

```
double faire_moyenne()
{
    return faire_moyenne_aux(0, 0);
}

double faire_moyenne_aux(double somme, int n)
{
    int terme = -1;

    printf("Entier positif :");
    scanf("%d", &terme);
    if (terme < 0) /* cas de base */
    {
        return somme / n; /* moyenne des termes precedents */
    }
    return faire_moyenne_aux(somme + terme, n + 1);
}
```