Algorithmique et Structures de données
Récursivité

Abdelkamel, Ben Ali

Université Echahid Hamma Lakhdar - El-Oued

Décembre 2020

Licence 2 d'informatique

Exemple 1

On appelle fonction récursive toute fonction qui, dans sa définition, fait appel à son propre identificateur. On peut ainsi définir la fonction somme comme suit :

$$somme(n) = \begin{cases} 0 \quad \text{si} \quad n = 0 \\ n + somme(n-1) \quad \text{si} \quad n > 0 \end{cases}$$

On pourrait donc décrire la fonction somme de la manière suivante :

$$somme(n) = \quad \text{si} \;\; n = 0 \;\; \text{alors} \;\; 0 \;\; \text{sinon} \;\; n + somme(n-1)$$

on a donc :

$$\begin{aligned} somme(4) &= 4 + somme(4-1) = 4 + somme(3) \\ &= 4 + 3 + somme(3-1) = 4 + 3 + somme(2) \\ &= 4 + 3 + 2 + somme(2-1) = 4 + 3 + 2 + somme(1) \\ &= 4 + 3 + 2 + 1 + somme(1-1) = 4 + 3 + 2 + 1 + somme(0) \\ &= 4 + 3 + 2 + 1 + 0 = 10 \end{aligned}$$

Ce qu'on traduira en C par :

```
void somme(int n)
{
    if (n == 0) return 0;
    else return (n + somme(n - 1));
}
```

Exemple 2

De la même manière on pourrait définir la multiplication de x par n ($n \in \mathbb{N}_+$) de la manière suivante :

$$multiplication(x, n) = \begin{cases} 0 \quad \text{si} \quad n = 0 \\ x + multiplication(x, n-1) \quad \text{si} \quad n > 0 \end{cases}$$

on aurait donc :

$$\begin{aligned} multi(4,3) &= 4 + multi(4, 3-1) = 4 + multi(4,2) \\ &= 4 + 4 + multi(4, 2-1) = 8 + multi(4,1) \\ &= 8 + 4 + multi(4, 1-1) = 12 + multi(4,0) \\ &= 12 + 0 = 12 \end{aligned}$$

Ce qu'on traduira en C par :

```
void multi(int x, int n)
{
    if (n == 0) return 0;
    else return (x + multi(x, n - 1));
}
```

Terminaison

- On dira qu'une fonction f et un argument x que $f(x)$ termine si ce calcul nécessite un nombre fini d'opérations élémentaires.
- Prouver la terminaison d'une fonction récursive consiste à montrer que l'on arrive toujours au traitement des cas de base en un nombre fini d'appels. Pour cela, on peut utiliser la notation intuitive suivante :
- *Une fonction récursive termine si on dispose d'une quantité liée à la fonction récursive et à ses arguments tels que* :
  - *Cette quantité évolue dans un ensemble ordonné qui ne comporte pas de suite infinie strictement décroissante.*
  - *A chaque appel de la fonction cette quantité décroît strictement.*
  - *La fonction récursive achève son calcul pour les cas de base.*

A l'endroit et à l'envers

Soit la fonction `compte_envers` définie par :

```
void compte_envers(int x)
{
     if (x < 0) {
          return;
     }
     else
     {
          printf("\nx = %d", x);
          compte_envers(x - 1);
     }
}
```

On remarque qu'à l'exécution de la fonction, l'impression se produit avant l'appel récursif. L'appel de fonction `compte_envers(4)` affichera donc successivement 4, 3, 2, 1 puis 0.

A l'endroit et à l'envers (2)

Par contre, si on modifie la fonction en plaçant l'impression après l'appel récursif :

```
void compte_endroit(int x)
{
      if (x < 0) {
            return;
      }
      else
      {
            compte_endroit(x - 1);
            printf("\nx = %d", x);
      }
}
```

On remarque qu'à l'exécution de la fonction, l'impression se produit après l'appel récursif. L'appel de fonction `compte_endroit(4)` affichera donc successivement 0, 1, 2, 3 puis 4.

Récursivité croisée ou mutuelle

Les fonctions peuvent être mutuellement récursives, par exemple :

```
void tic()
{
     printf("tic"); tac();
}
```

```
void tacc()
{
     printf("tac"); tic();
}
```

Ou encore

```
int pair(int x)
{
     if (x == 0) return 1;
     else return impair(x - 1);
}
```

```
int impair(int x)
{
     if (x == 0) return 1;
     else return pair(x - 1);
}
```

Exercices

1. Calculer le nième nombre de Lucas défini par :
   $L_1 = 1$,
   $L_2 = 3$;
   $L_n = L_{n-1} + L_{n-2}$

2. Calculer Le P.G.C.D (Plus Grand Commun Diviseur) de deux nombres entiers en utilisant l'algorithme d'Euclide :

$$PGCD(a,b) = \begin{cases} a & \text{si} \quad a = b \\ PGCD(a-b,a) & \text{si} \quad a > b \\ PGCD(a,b-a) & \text{si} \quad a < b \end{cases}$$