Algorithmique et programmation – Cours 7 et 8. Première partie, fonctions récursives et révisions.

Pierre Boudes

22 novembre 2012

Contenu du contrôle

- Au dernier contrôle, vous aurez le mêmes types d'exercices qu'au premier contrôle, sans QCM, avec différents types de données (`int`, `char`, `double`, tableaux, `struct`) et des fonctions.
  - Commenter, compléter un programme (cours)
  - utiliser les principales structures de contrôle : if, for, while
  - un exercice porte sur la structuration de données (struct ou tableaux)
- le programme à modifier/compléter et l'exercice sur les données sont forcément avec fonctions
- Gérez votre temps, apprenez à lire un sujet et ne rien oublier.
  Nous testons vos acquis, allez à l'essentiel !

Structure et contenu d'un programme C

```c
/* Declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */
...

/* Declaration des constantes et types utilisateurs */
...

/* Declaration des fonctions utilisateurs */
...

/* Fonction principale */
int main()
{
    /* Declaration et initialisation des variables */
    ...
    /* valeur fonction */
    return EXIT_SUCCESS;
}

/* Definitions des fonctions utilisateurs */
...
```

Directives préprocesseur

```
/* Declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */
#include <math.h> /* pow, sqrt */ /* bibliotheque */

/* Declaration des constantes et types utilisateurs */
#define N 5 /* constante symbolique */
#define TRUE 1
#define FALSE 0

/* Declaration des fonctions utilisateurs */
...

/* Fonction principale */
int main()
{
    /* Declaration et initialisation des variables */
    int donnee[N];
    ...
    /* valeur fonction */
    return EXIT_SUCCESS;
}
```

Types utilisateurs struct

```
...
/* Declaration des constantes et types utilisateurs */
...
typedef struct paire_s {
    int g; /* gauche */
    int d; /* droite */
} paire_t ;

/* Declaration des fonctions utilisateurs */
...

/* Fonction principale */
int main()
{
    /* Declaration et initialisation des variables */
    struct paire_s meschaussures = {37, 44};
    ...
    /* valeur fonction */
    return EXIT_SUCCESS;
}

/* Definitions des fonctions utilisateurs */
```

Fonctions : déclarations (type), appels, définitions

```
...
/* Declaration des fonctions utilisateurs */
int factorielle(int n);                    /*     Z -> Z     */
int pgcd(int a, int b);                    /* Z x Z -> Z     */
double neper(int ordre);                   /*     Z -> R     */
void afficher_paire( paire_t  x);          /* paire -> rien */
int saisie_choix();                       /*  rien -> Z     */

/* Fonction principale */
int main()
{
    ...
    afficher_paire(meschaussures); /* appel */
    ...
}

/* Definitions des fonctions utilisateurs */
double neper(int n) /* definition de la fonction neper */
{
    ...
    somme = somme + 1.0 / factorielle(k); /* appel */
    ...
```

Fonctions récursives

```
...
/* Declaration des fonctions utilisateurs */
...
double neper(int ordre);                    /*      Z -> R     */
...

/* Fonction principale */
int main()
{
    ...
}

/* Definitions des fonctions utilisateurs */
double neper(int n) /* definition de la fonction neper */
{
    if (n > 1)
    {
        return 1.0 / factorielle(n)  + neper(n - 1); /* appel recur
    ...
```

Pointeurs, paramètres par adresses

```
...
/* Declaration des fonctions utilisateurs */
...
void echanger(int *a, int *b);
double sommer_tableau(double t[], int taille);

/* Fonction principale */
int main()
{
    int x = 3;
    int y = 5;
    double t[3] = {3.4, 1.2, 1.1};
    double somme;
    echanger(&x, &y); /* x = 5 et y = 3 */
    somme = sommer_tableau(t, 3);
    ...
}
...
```

Rappel sur les fonctions en C

Utilisation des fonctions :

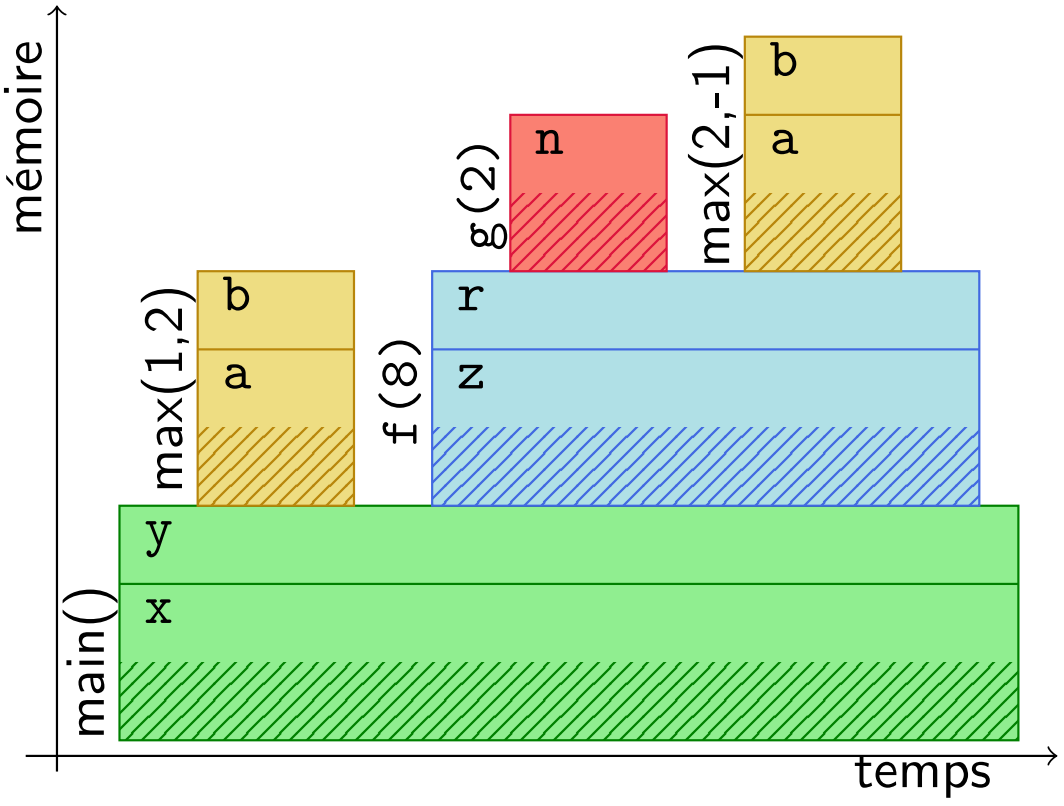
- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs, **espace mémoire**)

Nous reviendrons sur cette question d'espace mémoire un peu plus tard aujourd'hui.

Pile d'appel

On parle de pile d'appel car les appels de fonctions s'empilent... *comme sur une pile d'assiettes.*

Peut-on avoir deux éléments identiques dans la pile ? (La même fonction avec des paramètres éventuellement différents)



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.

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`

- Comment obtenir `1 2 3 4 5` ?

Exemples. Affichage à la remontée

Et avec le code :

```
 7  int factorielle(int n)
 8  {
 9      int res = 1;
10      printf("%d ", n);
11      if (n > 1) /* cas recursif */
12      {
13          res =  n * factorielle(n - 1);
14      }
15      printf("%d ", n);
16      return res;
17  }
```

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` ?

Exemple. 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.

```
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);
}
```