Année 2005–2006 — Session de Mai 2006

Parcours : INF2, MAT2 **Date :** 5 Septembre 2006
Code UE : INF101 **Durée :** 1h30
Corrigé de : Initiation à la programmation

Exercice 1

1. **Vrai**, car l'affectation modifie la valeur de la variable `a` (voir transparent 78).
2. **Vrai**, voir transparents 150–151.
3. **Faux**, le membre gauche d'une affectation ne peut pas être un identificateur de tableau. Pour recopier le contenu du tableau `v` dans le tableau `u`, on peut soit recopier case par case en utilisant une boucle, soit en utiliser des fonctions de bibliothèque de la famille `memcpy` (non vues en cours, voir page de manuel).
4. **Vrai**, `p` est un identificateur de tableau, il ne peut donc pas être membre gauche d'une affectation (voir aussi transparents 150–151).
5. **Faux**, on peut écrire après la première case contenant `'\0'`, si la mémoire a été correctement allouée, mais ce que l'on y écrit ne fait pas partie de la chaîne de caractères qui débute à la première case du tableau. Cette chaîne s'arrête au premier `'\0'`. Par exemple, si un tableau `tab` contient, à partir de l'indice 0, successivement les caractères `'a'`, `'b'`, `'c'`, `'\0'`, `'\0'`, alors après l'affectation `tab[4]='d'`, le tableau contiendra successivement les caractères `'a'`, `'b'`, `'c'`, `'\0'`, `'d'`. Il représentera toujours la chaîne `"abc"`, comme on peut le vérifier par l'instruction `printf("%s\n", tab);`.
6. **Faux**, voir transparent 80 (cela dépend du compilateur).
7. **Faux**, la définition de la fonction `main` est obligatoire mais elle se trouve dans un fichier source, ou plus rarement dans une bibliothèque. Pour le contenu des fichiers d'en-tête, cf. transparents 129–131.
8. **Faux**, l'appel récursif doit être : `calcul_suite(a, n-1)` , sinon pour `n` positif on n'atteint jamais la valeur `0` qui permet de terminer la séquence d'appels récursifs.

Exercice 2

```c
#include <string.h>


void supprime_dernier_car(char *s)
{
   int longueur = strlen(s);

   if (longueur == 0)
      return;

   s[longueur - 1] = '\0';
}


void supprime_car(int pos, char *s)
{
   for (int i = pos; i < strlen(s); i++)
      s[i] = s[i+1];
}
```

```c


int position_car(char c, char *s)
{
   for (int i = 0; s[i] != '\0'; i++)
      if (s[i] == c)
         return i;

   return -1;
}



void supprime(char c, char *s)
{
   for (int i = position_car(c,s);
        i != -1;
        i = position_car(c,s))
      supprime_car(i, s);
}
```

Exercice 3

```c
#include <stdio.h>
#include <stdlib.h>
#include "inf101.h"

// Declaration obligatoire avant leur definition
// des fonctions u et v, mutuellement recursives.

long u(int);
long v(int);

long u(int n)
{
   if (n == 0)
      return 1;

   return 2*u(n-1) + v(n-1) ;
}

long v(int n)
{
   if (n == 0)
      return 2;

   return u(n-1) - v(n-1)   ;
}

void affiche(long t[], int n)
{
   for (int i = 0; i < n; ++i)
      printf("%ld\t", t[i]);

   printf("\n");
}

void usage(char *s)
{
   fprintf(stderr, "Usage: %s.\n", argv[0])
}
```

```c
int
main(int argc, char *argv[])
{
   if (argc != 1)
   {
      usage(argv[0]);
      return EXIT_FAILURE;
   }

   printf("Entrez un entier positif\n");
   int n = lire_entier();

   long u[n], v[n];

   u[0] = 1;
   v[0] = 2;
   for (int i = 1; i < n; ++i)
   {
      u[i] = 2*u[i-1] + v[i-1];
      v[i] = u[i-1] - v[i-1];
   }

   printf("u : ");
   affiche(u, n);
   printf("v : ");
   affiche(v, n);

   return EXIT_SUCCESS;
}
```

Exercice 4

```c
int nombre_elements_communs(int *tab1, int *tab2, int n1, int n2)
{
   int cpt = 0;

   for (int i = 0; i < n1 ; i++)
      for (int j = 0; j < n2 ; j++)
         if (tab1[i] == tab2[j])
         {
            cpt++;
            break; // fait sortir de la boucle for (int j...), pour
                   // compter une seule fois la valeur tab1[i] meme si
                   // elle apparait plusieurs fois dans tab2.
         }

   return cpt;
}
```

Exercice 5

```c
#include <stdbool.h>
#include <string.h>

bool est_carre(char *s)
{
  int longueur = strlen(s);

  if (longueur % 2)
     return false;

  int d = longueur/2;
  for (int i = 0; i < d ; i++)
     if (s[i] != s[i+d])
        return false;

  return true;
}
```

Exercice 6 L'état de la mémoire avant l'appel `printf()` est représenté sur la figure 1. Le source est rappelé ci-dessous, et le numéro de la ligne ayant créé un lien entre *pointee* et *pointeur* est donné entre parenthèses. Par exemple, l'instruction de la ligne 11, `x1[0] = &t;`, provoque le lien marqué (11) sur la figure.

L'instruction 12 provoque l'affectation de la valeur `1` à l'emplacement correspondant à la variable `t`, ce qui écrase la précédente valeur (affectation de la ligne 9).

```c
1   #include <stdio.h>
2
3   int *x1[1];
4
5   int main(void)
6   {
7      int **p, *s, t;
8
9      t = 5;
10     p = x1;
11     x1[0] = &t;
12     *x1[0] = 1;
13     s = *p;
14
15     printf("%d %d %d\n", **p, *s, t);
16
17     return EXIT_SUCCESS;
18  }
```

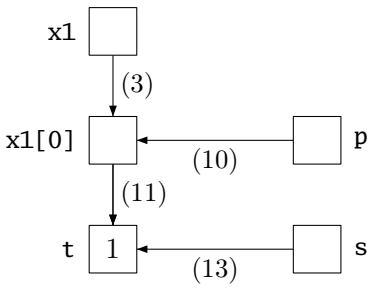

FIG. 1 – État de la mémoire avant l'appel `printf()`.

L'affichage obtenu est `1 1 1`.