MP2I
2021/2022

Informatique
Correction DS1
2 octobre 2021

CIV

1 Reprise des TPs - 10pts

1.1 OCaml - 5pts

Q01

```
let rec pow2 n =
    if n = 0
    then 1
    else 2 * pow2 (n-1)
```

Q02

```
let rec suplog2 n =
    if n = 0
    then 0
    else 1 + suplog2 (n/2)
```

Q03

```
let rec comb n k =
    if k > n || k < 0
    then 0
    else if k = n
    then 1
    else comb (n-1) (k-1) + comb (n-1) k
```

Q04

```
let rec trib n =
    if n <= 1
    then 0
    else if n = 2
    then 1
    else trib (n-1) - trib (n-2) * trib (n-3)
```

Q05

On a $C_0 = C_1 = C_2 = 1$ et si $n > 2$,

$$C_n = 1 + C_{n-1} + C_{n-2} + C_{n-3}$$

Q06

```
let rec applique l f =
    match l with
    | [] -> []
    | t::q -> f t :: applique q f
```

Q07

```
let rec nieme l n =
    match l, n with
    | [], _ -> None
    | t::_, 0 -> Some t
    | _::q, _ -> nieme q (n-1)
```

Q08

```
let rec pourtout l f =
    match l with
    | [] -> true
```

```
    | t::q -> f t && pourtout q f
```

1.2 C - 5pts

Q09

```
int suplog2(int n)
{
    int p = 1;
    int d = 0;
    while (n >= p)
    {
        p = p*2;
        d = d+1;
    }
    return d;
}
```

Q10

```
int factorielle(int n)
{
    int p = 1;
    for (int i = 1; i <= n; i++)
    {
        p = p*i;
    }
    return p;
}
```

Q11

```
int somme_cubes(int t[], int nb_elts)
{
    int s = 0;
    for(int i = 0; i < nb_elts; i++)
    {
        s = s + t[i]*t[i]*t[i];
    }
    return s;
}
```

Q12

```
void swap(int t[], int i, int j)
{
    int v = t[i];
    t[i] = t[j];
    t[j] = v;
}
```

Q13

```
int indice_minimum_sous_tableau(int t[], int i, int nb_elts)
{
    int val_min = t[i];
    int indice_min = i;

    for (int j = i+1; j < nb_elts; j++)
    {
        if (val_min > t[j])
        {
            val_min = t[j];
            indice_min = j;
        }
    }
```

```
    return indice_min;
}
```

Q14

```
void tri(int t[], int nb_elts)
{
    for (int i = 0; i < nb_elts; i++)
    {
        int j = indice_minimum_sous_tableau(t, i, nb_elts-i);
        swap(t, i, j);
    }
}
```

Q15 C'est un tri par sélection car on sélectionne le maximum des éléments non triés pour le placer à la bonne position.

Q16

```
int recherche_motif(char s[], char m[])
{
    int n = strlen(s);
    int p = strlen(m);
    for (int i = 0; i <= n-p; i++)
    {
        int j;
        for (j = 0; j < p; j++)
        {
            if (s[i+j] != m[j])
            {
                break;
            }
        }
        if (j == p)
        {
            return i;
        }
    }

    return -1;
}
```

2 Autour du crible d'Eratosthène - 2pts

Q17

```
void crible(int p[], int N)
{
    int i;
    p[0] = 0;
    p[1] = 0;
    for (i = 2; i <= N; i++)
    {
        p[i] = 1;
    }

    for (i = 2; i <= N; i++)
    {
        if (p[i] == 1)
        {
            int m = 2*i;
```

```
                while (m <= N)
                {
                    p[m] = 0;
                    m = m + i;
                }
            }
        }
}
```

3 Tours de Hanoi - 3pts

Q18 Uniquement pour $n = 1$ (ou 0 mais c'est un cas extrême) car si on au moins deux disques, seul le plus petit peut bouger entre les deux poteaux et on se retrouve bloqué.

Q19 On déplace les $n-1$ premiers disque sur le poteau du milieu en utilisant le poteau final comme espace temporaire. On se retrouve alors dans la configuration suivante : le premier poteau contient le plus grand disque, le deuxième les $n-1$ premiers, et le dernier est vide. On a place le plus grand disque sur le troisième poteau. On peut alors déplacer les $n-1$ disques du milieu sur le poteau final.
Le point clé à remarquer c'est qu'un poteau ait le plus grand disque ou pas dessus ne change rien vis-à-vis des $n-1$ disques restants : ils sont tous plus petits.

Q20
```
void hanoi(int n, int src, int dst, int inter)
{
    if (n != 0)
    {
        hanoi(n-1, src, inter, dst);
        deplace(src, dst);
        hanoi(n-1, inter, dst, src);
    }
}
```

Q21 $D_0 = 0$ car on ne fait rien et si $n > 0$, $D_n = 2D_{n-1} + 1$ car on a fait deux appels récursif et un déplacement. On a donc $D_n = 1 + 2 + \cdots + 2^{n-1} = 2^n - 1$.

4 Couvertures - 5pts

Q22 Dans le cas dégénéré $n = 0$ il n'y a qu'une couverture : c'est l'ensemble vide. Dans le csa $n = 1$ on ne peut placer qu'une tuile de taille 1.
Sinon on place une tuile de taille 1 et on couvre les $n-1$ restants ou on place une tuile de taille 2 et on couvre les $n-2$ restants.

```
let rec couvertures12 n =
    if n = 0 || n = 1
    then 1
    else couvertures12 (n-1) + couvertures12 (n-2)
```

Q23 Il s'agit de la suite de Fibonacci !

Q24 Pour couvrir un échiquier $2xn$ on a le choix sur la position du premier domino : vertical ou horizontal. Si on le place verticalement, il ne reste comme espace libre que de quoi placer un domino parallèle exactement à coté. Le choix que l'on fait est donc entre un domino horizontal ou deux dominos verticaux. C'est exactement le même choix qu'avec les tuiles précédentes. D'ailleurs on passe de l'un à l'autre en coupant l'échiquier en deux.

Q25 Si on colorie l'échiquier comme au échecs, chaque domino est à cheval sur une case blanche et une case noire. Il faut donc que le nombre de cases blanches de l'échiquier soit égal au nombre de cases noires.
Si $n = 2p + 1$, les lignes sont tantôt à p cases noires et $p + 1$ cases blanches, tantôt l'inverse. Pour que cela

s'équilibre, il faut que le nombre de lignes soit pair.
Si n est pair, l'équilibre est automatique.
La condition est donc d'avoir n ou m pair. Dans ce cas on peut découper l'échiquier en bandes de tailles $2xm$ ou $nx2$ et on sait que l'échiquier restant est couvrable (par exemple uniquement avec des dominos parallèles).
On a donc $n \times m$ admet une couverture $\iff$ nm pair (car n ou m pair $\iff$ nm pair).

Q26

```
let rec couverture tl n =
    let rec aux l = match l with
        | [] -> 0
        | t::q -> (if n>=t then couverture tl (n-t) else 0) + aux q
    in if n = 0 then 1 else aux tl;;
```

Q27

Ici on a quatre tuiles identiques de longueur 1, donc on a quatre choix par cases. Ainsi, s'il y a n cases alors on a 4^n couvertures distinctes.