TD 6 - Programmation impérative

1 Les incontournables

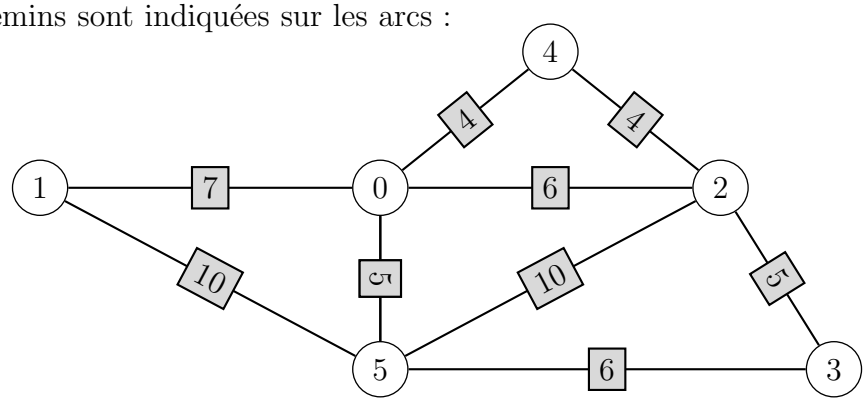
Exercice 1 Écrire une fonction `print_array` de type `int array -> unit` qui affiche les valeurs d'un tableau en les séparant par des "sauts de colonne" (caractère spécial `\t` dont le code ASCII vaut 9).

Exercice 2 Écrire une fonction `etendue` qui reçoit en entrée un tableau non vide d'entiers et renvoie l'écart entre la plus grande et la plus petite valeur de celui-ci.

Exercice 3 Programmer une fonction `pascal` de type `int -> int array` qui construit et renvoie à l'aide du triangle de Pascal, la n$^{\text{ème}}$ ligne du triangle. En affichant les étapes, on obtient alors quelque chose de la forme :

```
pascal 5;;
1        1
1        2        1
1        3        3        1
1        4        6        4        1
- : int array = [|1; 5; 10; 10; 5; 1|]
```

Exercice 4 Une carte a été modélisée par le graphe ci-dessous où les villes sont numérotées de 0 à 5 et les longueurs des chemins sont indiquées sur les arcs :



1. Déclarer la **matrice d'adjacence** de ce graphe, c'est-à- dire la matrice où les coefficients $a_{i,j}$ sont les distances entre la ville i et la ville j (0 si $i = j$). Lorsqu'il n'existe pas de chemin, on posera $a_{i,j} = -1$.
2. Écrire une fonction `voisins : int array array -> int -> bool array` qui reçoit une matrice d'adjacence et l'indice d'une ville et retourne un tableau de booléens indiquant les villes voisines :

```
voisins carte 0;;
- : bool array=[|false;true;true;false;true;true|]
```

3. Écrire une fonction `longueur : int array array -> int array -> int` qui reçoit une matrice d'adjacence et un tableau d'entiers indiquant une liste de villes et retourne la longueur totale du chemin reliant ces villes s'il en existe un et -1 sinon.

```
# longueur carte [|0;1;5|];;
- : int = 17
# longueur carte [|0;1;2|];;
- : int = -1
```

4. Proposez une version récursive n'utilisant pas de boucle.

Exercice 5 Écrire une fonction `find_chr` qui reçoit une chaine de caractères *txt* et un caractère *need* et retourne l'indice de la première occurrence de *need* dans *txt* le cas échéant et -1 sinon.
▷ Il existe déjà, dans le module `String`, une fonction `index` s c qui retourne la position de la première occurrence du caractère c dans la chaine s et déclenche l'erreur "Not_found" si celui-ci n'apparait pas, mais nous n'allons pas l'utiliser ici.

Exercice 6 Écrire une fonction `voyelles` qui reçoit une chaîne de caractères en minuscule et renvoie la même chaine où toutes les voyelles ont été écrites en majuscule.

2 Pour s'entrainer

Exercice 7 Écrire une fonction `modale` qui reçoit en entrée un tableau et renvoie la valeur (supposée unique) qui apparait le plus souvent dans celui-ci.

Exercice 8 Reprendre l'exercice 3 en utilisant un seul tableau !

Exercice 9 Écrire les fonctions :

1. `somme : int array array -> int` qui reçoit une matrice et retourne la somme des éléments.
2. `nbsup : int array array -> int -> int` qui reçoit une matrice et une valeur et renvoie le nombre d'éléments strictement supérieurs à cette valeur.
3. `transpose : 'a array array -> 'a array array` qui reçoit une matrice et renvoie la matrice transposée, c'est-à-dire la matrice où les lignes et les colonnes ont été inversées.

Exercice 10 La fonction **91 de McCarthy** est une fonction récursive définie pour $n \in \mathbb{N}$ par

$$f(n) = \begin{cases} n - 10 & \text{si } n > 100 \\ f(f(n + 11)) & \text{sinon.} \end{cases}$$

1. Programmer cette fonction.
2. Conjecturer ce qu'elle renvoie.
3. Le démontrer.

Exercice 11 Écrire une fonction `palindrome : string -> bool` : qui indique si une chaîne de caractères est un palindrome.

Correction 1 Cette fonction pourra vous être utile lors de vos TIPE pour débogguer vos programmes !

```
let print_array t =
   for i = 0 to Array.length t - 1 do
      print_int t.(i);
      print_string "\t";
   done;
   print_newline()
;;
```

Correction 2 En une boucle, on cherche le minimum et le maximum du tableau (On suppose que le tableau n'est pas vide)

```
let etendue v =
   let mini = ref v.(0) and maxi = ref v.(0) in
      for i = 0 to Array.length v - 1 do begin
         if v.(i) > !maxi then maxi := v.(i);
         if v.(i) < !mini then mini := v.(i);
      end done;
      !maxi - !mini
;;
```

Correction 3 En utilisant la formule du triangle de Pascal (noter que l'on remplit le tableau "à l'envers" pour ne pas écraser les valeurs) :

```
let pascal n =
   let rep = Array.make (n+1) 0 in
   rep.(0) <- 1;
   for ligne = 1 to n do
      for colonne = ligne downto 1 do
         rep.(colonne) <- rep.(colonne)+rep.(colonne-1)
      done;
      print_vect rep;
   done;
   rep
;;
```

Correction 4 1. Déclaration :

```
let carte = [| [|  0;  7;  6; -1;  4;  5 |];
               [|  7;  0; -1; -1; -1; 10 |];
               [|  6; -1;  0;  5;  4; 10 |];
               [| -1; -1;  5;  0; -1;  6 |];
               [|  4; -1;  4; -1;  0; -1 |];
               [|  5; 10; 10;  6; -1;  0 |] |];;
```

2. En réalisant une boucle pour tester tous les voisins potentiels :

```
let voisins c v =
   let n = Array.length c in
      let rep = Array.make n true in
         for i = 0 to n-1 do
            if c.(i).(v) < 1 then rep.(i) <- false
         done;
   rep
;;
```

On peut aussi utiliser la fonction `map` qui renvoie un nouveau tableau en appliquant une fonction donnée :

```
let voisins c v = Array.map (function x -> x <> -1) c.(v);;
```

3. On réalise une boucle sur le chemin, dès que l'on trouve un -1, inutile de continuer les calculs :

```
let longueur c chem =
   let l = ref 0 and n = Array.length chem in
      for i = 0 to n-2 do
         if !l <> -1 then
            let d = c.(chem.(i)).(chem.(i+1)) in
            if d = -1 then l := -1 else l := !l + d
      done;
   !l
;;
```

Un `while` est plus efficace, il permet de sortir dès qu'un arc n'est pas présent :

```
let longueur c chem =
   let l = ref 0 and n = Array.length chem and i = ref 0 in
      while !i < n-1 && c.(chem.(!i)).(chem.(!i+1)) <> -1 do
         l := !l + c.(chem.(!i)).(chem.(!i+1));
         i := !i + 1;
      done;
      if !i = n-1 then !l else -1
;;
```

4. A l'aide de la fonction `sub` (voir la documentation OCaml), on grappille le chemin à chaque fois que l'on avance. La difficulté ici est de ne pas ajouter le résultat lorsque la longueur trouvée est -1 :

```
let rec longueur c chem =
   if Array.length chem < 2
   then 0
   else let d = c.(chem.(0)).(chem.(1)) in
      if d = -1
      then -1
      else
         let lg = longueur c (Array.sub chem 1 (Array.length chem - 1)) in
         if lg = -1 then -1 else d +lg
;;
```

ou sans utiliser la fonction `sub` :

```
let longueur c chem =
   let rec longR = function
        -1 -> 0
     | deb -> let d = c.(chem.(deb)).(chem.(deb+1)) in
              if d = -1
              then -1
              else let lg = longR (deb-1) in
                 if lg = -1 then -1 else d + lg
   in longR (Array.length chem - 2)
;;
```

Correction 5 On parcourt la chaine jusqu'à trouver le caractère ou jusqu'à la fin si on ne le trouve pas. La double condition du while permet de sortir de la boucle dès que le caractère est trouvé :

```
let find_chr txt need =
  let ret = ref (-1) and i = ref 0  in
    while !i < String.length txt && !ret = -1 do
      if txt.[!i] = need then ret := !i;
      i := !i + 1;
    done;
  !ret;;
```

Correction 6 Une première solution récursive, car on ne peut pas modifier un caractère d'une chaîne :

```
let voyelle txt =
        let rec voyelleR txt i =
                if i = String.length txt
                then ""
                else match txt.[i] with
                        | 'a' -> "A" ^ voyelleR txt (i+1)
                        | 'e' -> "E" ^ voyelleR txt (i+1)
                        | 'i' -> "I" ^ voyelleR txt (i+1)
                        | 'o' -> "O" ^ voyelleR txt (i+1)
                        | 'u' -> "U" ^ voyelleR txt (i+1)
                        | 'y' -> "Y" ^ voyelleR txt (i+1)
                        | _ -> String.make 1 (txt.[i]) ^ voyelleR txt (i+1)
        in voyelleR txt 0
;;
```

Une autre solution sans fonction auxiliaire :

```
let rec voyelle txt =
  if txt = ""
  then ""
  else match txt.[0] with
      | 'a' -> "A" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'e' -> "E" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'i' -> "I" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'o' -> "O" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'u' -> "U" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | 'y' -> "Y" ^ voyelle (String.sub txt 1 (String.length txt - 1))
      | _ -> String.make 1 (txt.[0]) ^ voyelle (String.sub txt 1
                                                (String.length txt - 1))
;;
```

Voici une troisième solution utilisant une conversion de type `String -> Bytes`.

```
let voyelle txt =
   let b_txt = Bytes.of_string txt in
      for i = 0 to Bytes.length(b_txt) -1 do
         let c = Bytes.get b_txt i in
         if (c='a') || (c='e') ||(c='i') ||(c='o') ||(c='u')
         then Bytes.set b_txt i (char_of_int(int_of_char(c)-32));
      done;
   Bytes.to_string b_txt
;;
```

Pour tester si `c` est une voyelle, on peut utiliser la fonction `contains` du module `String`. De même pour passer en majuscule, on peut utiliser la fonction `uppercase_ascii` du module `Char` :

```
let voyelle txt =
   let b_txt = Bytes.of_string txt in
      for i = 0 to Bytes.length(b_txt) -1 do
         let c = Bytes.get b_txt i in
         if String.contains "aeiuoy" c
         then Bytes.set b_txt i (Char.uppercase_ascii c);
      done;
   Bytes.to_string b_txt
;;
```

Enfin, on peut se passer de la conversion de type en :

- Créant une fonction (`voyelle_maj : char -> char`) qui retourne le caractère donné en majuscule si c'est une voyelle et en minuscule sinon.
- Appliquant cette fonction à tous les caractères de la chaîne à l'aide de la fonction `map` du module `String`

```
let voyelles txt =
   let voyelle_maj c =
      if String.contains "aeiouy" c
      then Char.uppercase_ascii c
      else c
   in String.map voyelle_maj txt
;;
```

Correction 7 Voici une solution qui peut largement être améliorée : on crée une fonction auxiliaire `nb_fois` qui compte le nombre de fois où une valeur apparait.

```
let modale t =
        let nb_max = ref 0 and v_max = ref t.(0) and nb_fois t v =
                let nb = ref 0 in
                for i = 0 to Array.length t - 1 do
                        if t.(i) = v then nb := !nb + 1
                done;
                !nb in
        for i = 0 to Array.length t - 1 do
                let x = nb_fois t t.(i) in
                if x > !nb_max then begin
                        nb_max := x;
                        v_max := t.(i)
                end
        done;
        ! v_max;;
```

Correction 8 Déjà corrigé avec cette solution !

Correction 9

1. A l'aide d'un accumulateur `s` :

```
let somme t =
    let s = ref 0 and li = Array.length t and co = Array.length t.(0) in
        for l = 0 to li-1 do
            for c = 0 to co-1 do
                s := !s + t.(l).(c)
            done;
        done;
        !s;;
```

2. Ici, on peut forcer le type de v si on le souhaite (sinon, on a une fonction polymorphe) :

```
let nbsup t (v:int) =
  let nb = ref 0 and li = Array.length t and co = Array.length t.(0) in
    for l = 0 to li-1 do
      for c = 0 to co-1 do
        if t.(l).(c) > v then nb := !nb + 1
      done;
    done;
    !nb;;
```

3. On crée un tableau vide, que l'on remplit :

```
let transpose t =
    let li = Array.length t and co = Array.length t.(0) in
        let tr = Array.make_matrix co li t.(0).(0) in
            for l = 0 to li-1 do
                for c = 0 to co-1 do
                    tr.(c).(l) <- t.(l).(c)
                done;
            done;
        tr;;
```

Correction 10 1. Le code ne devrait pas poser de problème :

```
let rec mc = function
        | n when n > 100 -> n - 10
        | n -> mc ( mc(n+11) )
;;
```

2. Après quelques tests on conjecture que $f(n) = \begin{cases} n-10 \text{ si } n > 100 \\ 91, \text{ sinon} \end{cases}$

3. Le cas $n > 100$ est évident de par la définition. Pour les autres, voici une jolie preuve informatique :

```
for i = 0 to 100 do
  print_int i;
  print_string ":";
  print_int(mc i);
  print_newline()
done;;
```

Correction 11 Une première solution où l'on parcourt tout le mot :

```
let palindrome mot =
  let nb = ref 0 and l = String.length mot - 1 in
    for i = 0 to l / 2 do
      if mot.[i] <> mot.[l-i] then nb := !nb + 1;
    done;
    !nb = 0;;
```

Une seconde où l'on s'arrête dès que 2 lettres diffèrent :

```
let palindrome mot =
  let nb = ref true and l = String.length mot - 1 and i = ref 0 in
    while (!i <= l / 2) && !nb do
      if mot.[!i] <> mot.[l - !i] then nb := false;
      i := !i + 1;
    done;
    !nb;;
```