Correction du TP1 : Programmation récursive sur les listes

Programmation de certaines primitives sur les listes

1.

```
let rec appartient x l= match l with
  |[]->false
  |t::q->(t=x)||(appartient x q);;
```

2.

```
let rec occurence x l= match l with
  |[]->0
  |t::q->if (t=x) then
                  1+(occurence x q)
           else
                  occurence x q ;;
```

3.

```
let rec dernier l= match l with
  |[x]->x
  |t::q->dernier q;;
```

4.

```
let rec ajout_fin x l=match l with
  |[]->[x]
  |t::q->t::(ajout_fin x q);;
```

5.

```
let rec i_eme i l=let t::q=l in match i with
  |1->t
  |_->i_eme (i-1) q;;
```

6.

```
let rec supprime i l=let t::q=l in match i with
  |1->q
  |_->t::supprime (i-1) q;;
```

7.

```
let rec supprime_tout x l=match l with
  |[]->[]
  |t::q->if (t=x) then
                  supprime_tout x q
          else
                  t::(supprime_tout x q);;
```

8.

```
let rec insere x i l =match i with
  |1->x::l
  |_->let t::q=l in t::(insere x (i-1) q);;
```

9.

```
let rec concatene l1 l2=match l1 with
  |[]->l2
  |t::q->t::(concatene q l2);;
```

10.

```
let rec concatene_inverse l1 l2=match l1 with
  |[]->l2
  |t::q->concatene_inverse q (t::l2);;

let miroir l=concatene_inverse l [];;
```

Opérations « algébriques » sur les listes

11.

```
let rec addition l1 l2=match (l1,l2) with
  |([],_)->l2
  |(_,[])->l1
  |(t1::q1,t2::q2)->(t1+t2)::(addition q1 q2);;
```

12.

```
let rec fusion l1 l2=match(l1,l2) with
  |([],l2)->l2
  |(l1,[])->l1
  |(t1::q1,t2::q2)->if (t1<t2) then
                              t1::(fusion q1 l2)
                         else
                              t2::(fusion l1 q2);;
```

Algorithmes classiques sur les listes

13.

```
let rec maximum l=match l with
  |[x]->x
  |t::q->let m=(maximum q) in
              if (t>m) then
                      t
              else
                      m;;
```

14.

```
let rec second_maximum l=match l with
  |[x1;x2]->if (x1<x2) then x1 else x2
  |t::q->let m1=maximum q and m2= second_maximum q in
                  if (t < m2) then
                      m2
                  else
                      if (t<m1) then
                          t
                      else
                          m1;;
```

15.

```
let rec somme l=match l with
  |[]->0
  |t::q->t+(somme q);;
```

Algorithme de Hörner

16.

```
let rec evaluation l x=match l with
  |[]->0.
  |t::q->t +. x *. (evaluation q x);;
```

Coefficients binomiaux

17.

```
let rec binomial n p= match (n,p) with
 |(n,0)->1
 |(n,p) when (p>n)->0
 |_->(binomial (n-1) p)+(binomial (n-1) (p-1));;
```

18.

```
let rec ligne_suivante l=match l with
  |[1]->[1;1]
  |t1::(t2::q)->let (t3::q3)=(ligne_suivante (t2::q)) in
                          1::((t1+t2)::q3);;
```

19.

```
let rec pascal n= match n with
  |0->[1]
  |_->let l=(pascal (n-1)) in ligne_suivante l;;
```