Corrigé du TD$_6$

L'objet de ce TD est de définir un type "Polynôme", ainsi que les opérations de base sur ce type de données. On rappelle qu'un polynôme de degré n est en bijection avec le vecteur de ses $n+1$ coefficients.

```
1   program polynomes;
2
    const
4     DEGRE = 3;

6   type poly = array[0..DEGRE] of real;

8   procedure readPoly(var P: poly);
      var i: integer;
10    begin
        for i:=0 to DEGRE do
12        begin
            write('Terme de degre ', i, ': ');
14          readln(P[i]);
          end;
16    end;

18
    procedure writePoly(P: poly);
20    var i: integer;
      begin
22      for i:=DEGRE downto 0 do
          begin
24          write(P[i],' X^',i);
            if i<>0 then write(' + ');
26        end;
        writeln;
28    end;

30
    procedure sommePoly(P, Q: poly; var S: poly);
32    var i: integer;
      begin
34      for i:=0 to DEGRE do
          S[i] := P[i] + Q[i];
36    end;

38
    procedure lambdaPoly(lambda: real; P: poly; var R: poly);
40    var i: integer;
      begin
42      for i:=0 to DEGRE do
          R[i] := lambda * P[i];
44    end;

46
    procedure derivePoly(P: poly; var P1: poly);
48    var i: integer;
```

```
  begin
    for i:=1 to DEGRE do
      P1[i-1] := i * P[i];
    P1[DEGRE] := 0;
  end;


procedure primitivePoly(P: poly; var P1: poly);
  var i: integer;
  begin
    if P[DEGRE] = 0 then
      begin
        for i:=0 to DEGRE-1 do
          P1[i+1] := P[i]/(i+1);
        P1[0] := 0;
      end
    else writeln('Impossible de primitiver.');
  end;


function degrePoly(P: poly): integer;
  var i: integer;
  begin
    i:=0;
    while (P[i+1] <> 0) and (i<DEGRE) do i:=i+1;
    degrePoly := i;
  end;


procedure multPoly(P, Q: poly; var R: poly);
  var
    i, k: integer;
    somme: real;
  begin
    if degrePoly(P) + degrePoly(Q) <= DEGRE then
      begin
        for k:=0 to DEGRE do
          begin
            somme:=0;
            for i:=0 to k do somme:=somme + P[i]*Q[k-i];
            R[k] := somme;
          end;
      end
    else writeln('Impossible de multiplier.');
  end;


function evalPoly(P: poly; x0: real): real;
  var
    i, j: integer;
    val, puiss: real;
  begin
    val:=0;
    for i:=0 to DEGRE do
      begin
```

```
104           puiss:=1;                          (* calcul de *)
              for j:=1 to i do puiss:=puiss*x0;  (* x0 ^ i    *)
106           val:=val+P[i]*puiss;
            end;
108       evalPoly:=val;
      end;
110

112   function evalHorner(P: poly; x0: real): real;
        var
114       i: integer;
          val: real;
116
        begin
118       val:=0;
          for i:=DEGRE downto 0 do
120         begin
              val:=val*x0 + P[i];
122         end;
          evalHorner := val;
124     end;

126
      function integrePoly(P: poly; a,b: real): real;
128     var P1: poly;
        begin
130       primitivePoly(P, P1);
          integrePoly := evalHorner(P1, b) - evalHorner(P1, a);
132     end;

134


136
      var
138     P, Q, R: poly;
        lambda, a, b: real;
140
      begin
142     readpoly(P);
        writepoly(P);
144     readpoly(Q);
        writepoly(Q);
146     write('lambda: ');
        readln(lambda);
148     write('a: '); readln(a);
        write('b: '); readln(b);
150
        write('Somme: ');
152     sommepoly(P, Q, R);
        writepoly(R);
154
        write('Lambda: ');
156     lambdapoly(lambda, P, R);
        writepoly(R);
158
```

```
    write('Derive: ');
160 derivePoly(P, R);
    writepoly(R);
162
    write('Primitive: ');
164 primitivepoly(P, R);
    writepoly(R);
166
    write('Degre: ');
168 write(degrepoly(P));

170 write('Multiplication: ');
    multPoly(P, Q, R);
172 writepoly(R);

174 write('Evaluation naïve: ');
    writeln(evalpoly(P, lambda));
176
    write('Evaluation Hörner: ');
178 writeln(evalhorner(P, lambda));

180 write('Intégrale: ');
    writeln(integrepoly(P,a,b));
182 end.
```

Vous pouvez retrouver tous les énoncés et les corrigés des Travaux Dirigés en ligne :
`http://jacquetc.free.fr/carnot`

Bonnes Vacances !