Programmation Delphi : Algorithmes obligatoires

I^re B, 2006–07

Version 3.1 du 11 janvier 2007

Table des matières

1 Mathématiques élémentaires

1.1 Fonction « puissance à exposant naturel »

1.1.1 Version itérative

La fonction suivante calcule b^e pour un exposant e naturel. Le cas particulier $b = e = 0$ n'est pas traité correctement.

```
function PUISSANCE(BASE:extended; EXPO:integer):extended;
var I:integer;
    P:extended;
begin
  P:=1;
  for I:=1 to EXPO do
    P:=P*BASE;
  PUISSANCE:=P
end;
```

1.1.2 Version récursive

La fonction suivante calcule b^e pour un exposant e naturel. Le cas particulier $b = e = 0$ n'est pas traité correctement.

```
function PUISSANCE(BASE:extended; EXPO:integer):extended;
begin
  if EXPO=0 then
    PUISSANCE:=1
  else
    PUISSANCE:=BASE*PUISSANCE(BASE,EXPO-1)
end;
```

1.2 Fonction « puissance rapide à exposant naturel »

```
function PUISSRAPID(BASE:extended; EXPO:integer):extended;
begin
  if EXPO=0 then
    PUISSRAPID:=1
  else if EXPO mod 2 = 0 then
    PUISSRAPID:=PUISSRAPID(BASE*BASE,EXPO div 2)
  else
    PUISSRAPID:=BASE*PUISSRAPID(BASE,EXPO-1)
end;
```

1.3 Fonction « factorielle »

Les fonctions suivantes calculent $n!$ pour n naturel.

1.3.1 Version itérative

```
function FACTORIELLE(N:integer):integer;
var I:integer;
    FACT:integer;
begin
  FACT:=1;
  for I:=2 to N do
    FACT:=FACT*I;
  FACTORIELLE:=FACT
end;
```

1.3.2 Version récursive

```
function FACTORIELLE(N:integer):integer;
begin
  if N<2 then
    FACTORIELLE:=1
  else
    FACTORIELLE:=N*FACTORIELLE(N-1)
end;
```

1.4 Fonction « pgcd »

Les fonctions suivantes déterminent le pgcd de deux nombres naturels non nuls.

1.4.1 Algorithme d'Euclide par soustraction

```
function EUCLIDE_DIFF(A,B:integer):integer;
begin
  while A<>B do
    if A>B then
      A:=A-B
    else
      B:=B-A;
  EUCLIDE_DIFF:=A
end;
```

1.4.2 Algorithme d'Euclide par division

```
function EUCLIDE_DIVI(A,B:integer):integer;
var C:integer;
begin
  while B>0 do begin
    C:=A mod B;
    A:=B;
    B:=C
  end;
  EUCLIDE_DIVI:=A
end;
```

1.5 Nombre premier ?

```
function PREMIER(N:integer):boolean;
var I:integer;
    PRIM:boolean;
begin
  if N<2 then
    PREMIER:=false
  else if N=2 then
    PREMIER:=true
  else if N mod 2 = 0 then
    PREMIER:=false
  else begin
    I:=3;
    PRIM:=true;
    while (I*I<=N) and PRIM do
      if N mod I = 0 then PRIM:=false
      else I:=I+2;
    PREMIER:=PRIM
  end
end;
```

2 Polynômes

Le type POLY représente des polynômes à coefficients réels et de degré inférieur ou égal à 100.

```
type POLY = record
    C:array[0..100] of extended;
    D:integer
  end;
```

2.1 Addition

```
function SOMME(var A,B:POLY):POLY;
var I:integer;
    S:POLY;
begin
  if A.D<=B.D then begin
    S.D:=B.D;
    for I:=0 to A.D do
      S.C[I]:=A.C[I]+B.C[I];
    for I:=A.D+1 to B.D do
      S.C[I]:=B.C[I]
  end
  else begin
    S.D:=A.D;
    for I:=0 to B.D do
      S.C[I]:=A.C[I]+B.C[I];
    for I:=B.D+1 to A.D do
      S.C[I]:=A.C[I]
  end;
  while (S.D>0) and (S.C[S.D]=0) do
    S.D:=S.D-1;
  SOMME:=S
end;
```

2.2 Multiplication

```
function PRODUIT(var A,B:POLY):POLY;
var I,J:integer;
    P:POLY;
begin
  if ((A.D=0) and (A.C[0]=0)) or ((B.D=0) and (B.C[0]=0)) then begin
    P.D:=0;
    P.C[0]:=0
  end
  else begin
    P.D:=A.D+B.D;
    for I:=0 to P.D do
      P.C[I]:=0;
    for I:=0 to A.D do
      for J:=0 to B.D do
        P.C[I+J]:=P.C[I+J]+A.C[I]*B.C[J]
  end;
  PRODUIT:=P
end;
```

2.3 Dérivation

```
function DERIVEE(var A:POLY):POLY;
var I:integer;
    DER:POLY;
begin
  for I:=1 to A.D do
    DER.C[I-1]:=A.C[I]*I;
  if A.D=0 then begin
    DER.D:=0;
    DER.C[0]:=0
  end
  else
    DER.D:=A.D-1;
  DERIVEE:=DER
end;
```

2.4 Intégration

La procédure donne la primitive à terme constant 0.

```
function PRIMITIVE(var A:POLY):POLY;
var I:integer;
    PRIM:POLY;
begin
  PRIM.C[0]:=0;
  for I:=0 to A.D do
    PRIM.C[I+1]:=A.C[I]/(I+1);
  if (A.D=0) and (A.C[0]=0) then
    PRIM.D:=0
  else
    PRIM.D:=A.D+1;
  PRIMITIVE:=PRIM
end;
```

2.5 Schéma de Horner

```
function HORNER(var A:POLY; X:extended):extended;
var I:integer;
    PX:extended;
begin
  PX:=A.C[A.D];
  for I:=A.D-1 downto 0 do
    PX:=PX*X+A.C[I];
  HORNER:=PX
end;
```

3 Algorithmes de tri

Les algorithmes de cette section réalisent un tri *lexicographique*. La procédure `ECHANGE` réalise l'échange de deux éléments d'une liste.

```
1  procedure ECHANGE(var LISTE:TListBox; I,J:integer);
2  var AUX:string;
3  begin
4    AUX:=LISTE.Items[I];
5    LISTE.Items[I]:=LISTE.Items[J];
6    LISTE.Items[J]:=AUX
7  end;
```

3.1 Tri par sélection

3.1.1 Version récursive

```
1  procedure TRI_SELECTION_R(var LISTE:TListBox; DEBUT:integer);
2  var J,MIN:integer;
3  begin
4    MIN:=DEBUT;
5    for J:=DEBUT+1 to LISTE.Items.Count-1 do
6      if LISTE.Items[J]<LISTE.Items[MIN] then
7        MIN:=J;
8    ECHANGE(LISTE,DEBUT,MIN);
9    if DEBUT<LISTE.Items.Count-2 then
10     TRI_SELECTION_R(LISTE,DEBUT+1)
11 end;
```

3.1.2 Version itérative

```
1  procedure TRI_SELECTION_I(var LISTE:TListBox);
2  var I,J,MIN:integer;
3  begin
4    for I:=0 to LISTE.Items.Count-2 do begin
5      MIN:=I;
6      for J:=I+1 to LISTE.Items.Count-1 do
7        if LISTE.Items[J]<LISTE.Items[MIN] then
8          MIN:=J;
9      ECHANGE(LISTE,I,MIN)
10   end
11 end;
```

3.2 Tri par insertion

3.2.1 Version récursive

```
1  procedure TRI_INSERTION_R(var LISTE:TListBox; G,D:integer);
2  var J:integer;
3      CANDIDAT:string;
4  begin
5    if G<D then begin
6      TRI_INSERTION_R(LISTE,G,D-1);
7      CANDIDAT:=LISTE.Items[D];
8      J:=D;
9      while (J>G) and (LISTE.Items[J-1] > CANDIDAT) do begin
10       LISTE.Items[J]:=LISTE.Items[J-1];
11       J:=J-1
12     end;
13     if J<D then
14       LISTE.Items[J]:=CANDIDAT
15   end
16 end;
```

3.2.2 Version itérative

```
1  procedure TRI_INSERTION_I(var LISTE:TListBox);
2  var I,J:integer;
3      CANDIDAT:string;
4  begin
5    for I:= 1 to LISTE.Items.Count-1 do begin
6      CANDIDAT:=LISTE.Items[I];
7      J:=I;
8      while (J>0) and (LISTE.Items[J-1] > CANDIDAT) do begin
9        LISTE.Items[J]:=LISTE.Items[J-1];
10       J:=J-1
11     end;
12     if J<I then
13       LISTE.Items[J]:=CANDIDAT
14   end
15 end;
```

3.3 Tri rapide (*Quicksort*)

3.3.1 Version récursive

```
1  procedure TRI_RAPIDE_R(var LISTE:TListBox; G,D:integer);
2  var I:integer;
3  begin
4    if G<D then begin
5      I:=DIVISION(LISTE,G,D);
6      TRI_RAPIDE_R(LISTE,G,I-1);
7      TRI_RAPIDE_R(LISTE,I+1,D)
8    end
9  end;
```

3.3.2 Fonction auxiliaire « division »

```
1  function DIVISION(var LISTE:TListBox; G,D:integer):integer;
2  var I,J:integer;
3      CANDIDAT:string;
4  begin
5    CANDIDAT:=LISTE.Items[D];
6    J:=D-1;
7    I:=G;
8    while I<=J do
9      if LISTE.Items[I]<CANDIDAT then
10       I:=I+1
11     else if LISTE.Items[J]>CANDIDAT then
12       J:=J-1
13     else begin
14       ECHANGE(LISTE,I,J);
15       I:=I+1;
16       J:=J-1
17     end;
18   ECHANGE(LISTE,I,D);
19   DIVISION:=I
20 end;
```

4 Algorithme de recherche

4.1 Recherche séquentielle

```
1  function RECHERCHE_SEQ(LISTE:TListBox; CLE:string):integer;
2  var I:integer;
3  begin
4    I:=0;
5    while (I<LISTE.Items.Count) and (LISTE.Items[I]<>CLE) do
6      I:=I+1;
7    if I<LISTE.Items.Count then
8      RECHERCHE_SEQ:=I
9    else
10     RECHERCHE_SEQ:=-1
11 end;
```

4.2 Recherche dichotomique

4.2.1 Version récursive

```
1  function DICHO_R(LISTE:TListBox; CLE:string; G,D:integer):integer;
2  var MILIEU:integer;
3  begin
4    if G>D then
5      DICHO_R:=-1
6    else begin
7      MILIEU:=(G+D) div 2;
8      if LISTE.Items[MILIEU]=CLE then
9        DICHO_R:=MILIEU
10     else if CLE<LISTE.Items[MILIEU] then
11       DICHO_R:=DICHO_R(LISTE,CLE,G,MILIEU-1)
12     else
13       DICHO_R:=DICHO_R(LISTE,CLE,MILIEU+1,D)
14   end
15 end;
```

4.2.2 Version itérative

```
function DICHO_I(LISTE:TListBox; CLE:string):integer;
var MILIEU,G,D:integer;
begin
  G:=0;
  D:=LISTE.Items.Count-1;
  MILIEU:=(G+D) div 2;
  while (CLE<>LISTE.Items[MILIEU]) and (G<=D) do begin
    if CLE<LISTE.Items[MILIEU] then
      D:=MILIEU-1
    else
      G:=MILIEU+1;
    MILIEU:=(G+D) div 2
  end;
  if CLE=LISTE.Items[MILIEU] then
    DICHO_I:=MILIEU
  else
    DICHO_I:=-1
end;
```

4.3 Fréquence d'un élément dans une liste

```
function FREQUENCE(LISTE:TListBox; CLE:string):integer;
var I,FREQ:integer;
begin
  FREQ:=0;
  for I:=0 to LISTE.Items.Count-1 do
    if LISTE.Items[I]=CLE then
      FREQ:=FREQ+1;
  FREQUENCE:=FREQ
end;
```

4.4 Minimum d'une liste d'entiers non vide

```
function MINIMUM(LISTE:TListBox):integer;
var I,MINI:integer;
begin
  MINI:=StrToInt(LISTE.Items[0]);
  for I:=1 to LISTE.Items.Count-1 do
    if StrToInt(LISTE.Items[I])<MINI then
      MINI:=StrToInt(LISTE.Items[I]);
  MINIMUM:=MINI
end;
```

4.5 Maximum d'une liste d'entiers non vide

```
1 function MAXIMUM(LISTE:TListBox):integer;
2 var I,MAXI:integer;
3 begin
4   MAXI:=StrToInt(LISTE.Items[0]);
5   for I:=1 to LISTE.Items.Count-1 do
6     if StrToInt(LISTE.Items[I])>MAXI then
7       MAXI:=StrToInt(LISTE.Items[I]);
8   MAXIMUM:=MAXI
9 end;
```