12　あみだくじ 2

前回のあみだくじの問題は橋桁がたくさんあったので簡単にできたでしょう。実は，道が n 本のとき橋桁は ${}_nC_2 = n(n-1)/2$ 個あれば十分です。なぜかというと，橋を 1 つ架けることによって逆順になっているペアを交換することができます。上段のデータが完全に逆順になっている場合で，ペア全体である ${}_nC_2$ 個の橋を架けて正順にすることができます。この ${}_nC_2$ 個の橋桁を準備しておいて，その中の必要な箇所だけ橋を架ければよいのです。

12.1　橋桁を最少にする

完全に逆順にするための ${}_nC_2$ 個の橋桁だけに制限したもの 3 種類を選べるようにします。

12.1.1　スタイル選択ラジオグループ

RadioGroup をメインパネルの中に置く。

Align	alTop
Name	RadioGroupStyle
Caption	スタイル
Items	段数最少スタイル
	挿入整列法スタイル
	泡整列法スタイル
ItemIndex	0

OnClick ハンドラー

```
procedure TFormAmida.RadioGroupStyleClick(Sender: TObject);
begin
  Amida.Style := RadioGroupStyle.ItemIndex;
  Amida.SyokiSettei;
end; (* RadioGroupStyleClick *)
```

12.1.2　TAmida.SyokiSettei を変更

```
procedure TAmida.SyokiSettei;
          (* 横線の状況を初期化する *)
var
  Kawa : 1..TateSuu+1;
  Yoko : 1..YokoSuu+1;
  KawaStart : TKawaNo;
  KawaEnd   : TKawaNo;
begin
  Canvas.FillRect(Canvas.ClipRect);
  // どこにも橋も桁もない状態にする
  for Kawa := 1 to TateSuu-1 do
    for Yoko := 1 to YokoSuu do
      Hasi[Kawa,Yoko] := ysNone;
  // スタイルに従って桁を作る
  case Style of
```

```
      0 : begin // 段数最少
            for Yoko := 1 to TateSuu do
              begin
                Kawa := 2-Yoko mod 2;
                repeat
                  Hasi[Kawa,Yoko] := ysKeta;
                  Inc(Kawa,2);
                until Kawa >= TateSuu;
              end;
          end;
      1 : begin // 挿入整列法
            Yoko := 1;
            for KawaStart := 1 to TateSuu-1 do
             for Kawa := KawaStart downto 1 do
               begin
                 Hasi[Kawa,Yoko] := ysKeta;
                 Inc(Yoko);
               end;
          end;
      2 : begin // 泡整列法
            Yoko := 1;
            for KawaEnd := TateSuu-1 downto 1 do
              for Kawa := 1 to KawaEnd do
               begin
                 Hasi[Kawa,Yoko] := ysKeta;
                 Inc(Yoko);
               end;
          end;
    end;
    // データを初期状態に戻す
    Genzai := Hajime;
    LastYoko := 1;
    AmidaWoKaku;
  end; (* SyokiSettei *)
```

12.1.3　実行

スタイル 1（挿入整列法）と 2（泡整列法）は橋を架け終わった後のあみだくじは同じものができます。橋桁が右の川から順に 1 個，2 個，3 個，··· と増えている形が同じですから。しかし，“上流から順に” という条件があるので，橋の架け方が違います。

パズルとしては，橋桁の数が減ったので難しくなったと思ったら，解が 1 通りしかないのでかえって易しくなりました。

12.1.4　問題

3 つのスタイルそれぞれについて，新しい問題をやって，あみだくじを完成させなさい。

3 つのスタイルのあみだくじを確実に完成させることができるようになってから，次に進んでください。

12.2　自動化

人が橋を架けるのではなく，コンピュータが自動的に架けるようにします。

12.2.1　実行ボタン

Button をメインパネルの中に置く。
OnClick ハンドラー

```
procedure TFormAmida.Button1Click(Sender: TObject);
begin
  Amida.Style := 3;
  Amida.SyokiSettei;
  case TButton(Sender).Tag of
    0 : Amida.Compact;
    1 : {Amida.InsertSort};
    2 : {Amida.BubbleSort};
    3 : {Amida.InsertSort2};
    4 : {Amida.BubbleSort2};
  end;
end; (* ButtonJidouClick *)
```

このボタンを［編集｜コピー］，4 回［編集｜貼付け］をして，上のイベントハンドラーを共通に持ったボタンを 5 つ作る。それぞれのプロパティを設定する。

Name	Caption	Tag
ButtonCompact	段数最少	0
ButtonInsert	挿入整列法	1
ButtonBubble	泡整列法	2
ButtonInsert2	挿入整列法改良版	3
ButtonBubble2	泡整列法改良版	4

複数のボタンで似たような操作をするとき，同じイベントハンドラーを使うと便利です。どのボタンがクリックされたかわかるように，Tag プロパティに異なる番号をつけておきます。

「Amida.Style := 3 と書いたけど，スタイルは 0,1,2 しかないよ。またミスプリント？」
ミスプリントではありません。スタイル 3 は橋桁をあらかじめ作つておきません。自動実行メソッドが両岸のデータを**比較**する時に橋桁を作り，**交換**する時に橋を架けます。

12.2.2　段数最少実行メソッド

```
procedure TAmida.Compact;
          (* 段数最少で整列する *)
var
  Kawa : 1..TateSuu+1;
  Yoko : 1..YokoSuu+1;
begin
  for Yoko := 1 to TateSuu do
    begin
      Kawa := 2-Yoko mod 2;
```

```
      repeat
        Hasi[Kawa,Yoko] := ysKeta;          // 桁を作る
        HasiWoKaku(Kawa,Yoko);
        if Genzai[Kawa] < Genzai[Kawa+1]    // 左が右より小さいなら
          then HasiWoKakeru(Kawa,Yoko);     // 橋を架ける（交換する）
        Inc(Kawa,2);
      until Kawa >= TateSuu;
    end;
end; (* Compact *)
```

12.2.3　実行

段数が少ないコンパクトなあみだくしができます。
道を 11 本（奇数本）にするとどんな形になりますか。

12.2.4　問題

Compact のプログラムは，SyokiSettei のスタイル 0 の場合の橋桁作成部分とほとんど同じです。挿入整列法，泡整列法を自動実行する Inseert，Bubble の各メソッドを追加して実行できるようにしなさい。

2 つともできたら，次に進んでください。

12.3　改良

自動実行して眺めると，さっき比較してからデータが変わってないのにまた比較しているという，無駄な比較（橋桁作成）をしていることに気がつくでしょう。

挿入法と泡法について無駄な比較を少なくするように改良します。

```
procedure TAmida.InsertSort2;
          (* 挿入法で整列する 改良版 *)
var
  Kawa : 0..TateSuu-1;
  Yoko : 1..YokoSuu+1;
  KawaStart : TKawaNo;
begin
  Yoko := 1;
  for KawaStart := 1 to TateSuu-1 do
    begin
      Kawa := KawaStart;
      repeat
        Hasi[Kawa,Yoko] := ysKeta;           // 桁を作る
        HasiWoKaku(Kawa,Yoko);
        if Genzai[Kawa] < Genzai[Kawa+1]     // 左が右より小さいなら
          then HasiWoKakeru(Kawa,Yoko)       // 橋を架ける（交換する）
          else Kawa := 1;                    // 最後まで調べたことにする
        Inc(Yoko);
        Dec(Kawa);
      until Kawa = 0;
    end;
end; (* InsertSort2 *)

procedure TAmida.BubbleSort2;
          (* 泡法で整列する 改良版 *)
var
  Kawa : TKawaNo;
  Yoko : 1..YokoSuu+1;
  KawaEnd   : 0..TateSuu-1;
  KawaStart : TKawaNo;
  KoukanFirst : TKawaNo;
  KoukanLast  : TKawaNo;
begin
  Yoko := 1;
  KawaStart := 1;
  KawaEnd := TateSuu-1;
  repeat
    KoukanFirst := TateSuu-1;
    KoukanLast  := 1;
    for Kawa := KawaStart to KawaEnd do
      begin
        Hasi[Kawa,Yoko] := ysKeta;           // 桁を作る
        HasiWoKaku(Kawa,Yoko);
        if Genzai[Kawa] < Genzai[Kawa+1]     // 左が右より小さいなら
          then begin
                 HasiWoKakeru(Kawa,Yoko);    // 橋を架ける（交換する）
                 if Kawa < KoukanFirst
                   then KoukanFirst := Kawa; // 最初に橋を架けた川を記憶する
                 KoukanLast := Kawa;         // 最後に橋を架けた川を記憶する
               end;
        Inc(Yoko);
```

```
      end;
    if KoukanFirst > 1
      then KawaStart := KoukanFirst-1;
    KawaEnd := KoukanLast-1;
  until KawaStart > KawaEnd;
end; (* BubbleSort2 *)
```

12.3.1　実行

泡法はすべての無駄な比較をなくすことはできません。プログラムも挿入法に比べると複雑です。しかし，名前が面白いせいか有名で，いろいろな試験によく出ます。

皆さんは挿入法をおぼえてください。情報処理関係の試験を受ける人は泡法も理解してください。