Aufgabe 2 - Liniendiagram

**Lösungsidee:**

Das Auslesen und speichern der gelesenen Werte wird vor aufrufen der Application durchgeführt. Danach wird der gelesene Titel für das neue Window verwendet und im Window die gelesenen Daten gespeichert. Im Konstruktor des Windows wird auch der höchste Wert ermittelt (hätte man auch schon in der Application machen können), da der sich beim neu zeichnen nie ändert.

Die gesamten Chart Kalkulationen und zeichnen passieren alles in der Draw Methode (mit mehr Zeitaufwand hätte man das auch aufteilen und optimieren können, da die Methode ziemlich groß geworden ist und teilweise redundanten Berechnungen ausgeführt werden), dafür wird anhand der Höhe und Breite des Fensters mit der Anzahl der Werten und dem höchsten Wert der Abstand der unterschiedlichen Werte am Screen berechnet damit danach die Achsen und Achsen Beschriftung so viel vom Screen einnimmt wie möglich. Nachdem werden die Werte eingetragen, dafür wird die y Position relativ zum Koordinaten Ursprung und Max Höhe gerechnet.

**Zeitaufwand: ~**2h

**Code:**

program LineChart;

uses

  MetaInfo, OSBridge,

  MLObj, MLWin, MLAppl, MLVect, MLInt, MLColl;

type

  ChartType = (Simple, VerticalLines, HorizontalLines, GridLines);

  ChartWindow = ^ChartWindowObj;

  ChartWindowObj = object(MLWindowObj)

    data: MLVector;

    cType: ChartType;

    maxVal: integer;

    constructor Init(title: STRING; data: MLVector);

    destructor Done; virtual;

    (\*overridden methods\*)

    procedure Open; virtual;

    procedure Redraw; virtual;

    procedure OnCommand(commandNr: INTEGER); virtual;

    (\*new methods\*)

    procedure DrawChart;

    procedure ChangeChartType(cType: ChartType);

  end; (\*OBJECT\*)

  ChartApplication= ^ChartApplicationObj;

  ChartApplicationObj = object(MLApplicationObj)

    title: STRING;

    data: MLVector;

    constructor Init(name, filename: STRING);

    destructor Done; virtual;

    (\*overridden methods\*)

    procedure OpenNewWindow; virtual;

    procedure BuildMenus; virtual;

    (\*new methods\*)

    procedure ExtractDataFromFile(filename: STRING);

  end; (\*OBJECT\*)

var

  (\*chart types:\*)

  simpleLinesCommand, verticalLinesCommand, horizontalLinesCommand, gridLinesCommand: INTEGER;

(\*=== ChartWindow ===\*)

function NewChartWindow(title: string; data: MLVector): ChartWindow;

var

  w: ChartWindow;

begin

  New(w, Init(title, data));

  NewChartWindow := w;

end; (\*NewChartWindow\*)

constructor ChartWindowObj.Init(title: STRING; data: MLVector);

var

  iterator: MLIterator;

  next: MLObject;

begin

  inherited Init(title);

  Register('ChartWindow', 'MLWindow');

  cType := Simple;

  self.data := data;

  maxVal := MLInteger(data^.GetAt(data^.Size))^.AsInteger;

  iterator := data^.NewIterator;

  next := iterator^.Next;

  maxVal := MLInteger(next)^.AsInteger;

  while next <> NIL do

  begin

    if (MLInteger(next)^.AsInteger > maxVal) then

      maxVal := MLInteger(next)^.AsInteger;

    next := iterator^.Next;

  end;

  maxVal := Round(maxVal / 100) + 1; // round up to nearest multiple of 100

  Dispose(iterator, Done);

end; (\*ChartWindowObj.Init\*)

destructor ChartWindowObj.Done;

begin

  Dispose(data, Done);

  inherited Done;

end; (\*ChartWindowObj.Done\*)

procedure ChartWindowObj.Open;

begin

  inherited Open;

  DrawChart;

end; (\*ChartWindowObj.Open\*)

procedure ChartWindowObj.Redraw;

begin

  DrawChart;

end; (\*ChartWindowObj.Redraw\*)

procedure ChartWindowObj.OnCommand(commandNr: INTEGER);

begin

  if commandNr = simpleLinesCommand then

    ChangeChartType(Simple)

  else if commandNr = verticalLinesCommand then

    ChangeChartType(VerticalLines)

  else if commandNr = horizontalLinesCommand then

    ChangeChartType(HorizontalLines)

  else if commandNr = gridLinesCommand then

    ChangeChartType(GridLines)

  else

    inherited OnCommand(commandNr);

end; (\*ChartWindowObj.OnCommand\*)

procedure ChartWindowObj.ChangeChartType(cType: ChartType);

begin

  DrawChart;

  self.cType := cType;

  DrawChart;

end; (\*ChartWindowObj.ChangeChartType\*)

procedure ChartWindowObj.DrawChart;

var

  currVal, horizontalGap, verticalGap, i: integer;

  origin, dummy1, dummy2: Point;

  curr, prev: Point;

  intStr: string;

begin

  horizontalGap := (Width - 80) div (data^.Size - 1);

  verticalGap := (Height - 50) div maxVal;

  origin.x := 50;

  origin.y := Height - 20;

  // draw axes

  dummy1.x := origin.x + horizontalGap \* (data^.Size - 1);

  dummy1.y := origin.y;

  DrawLine(origin, dummy1, 1);

  dummy1.x := origin.x;

  dummy1.y := origin.y - verticalGap \* maxVal;

  DrawLine(origin, dummy1, 1);

  // draw grid horizontal

  dummy1.x := origin.x - 30;

  dummy1.y := origin.y - 10;

  for i := 0 to maxVal do

  begin

    Str(i\*100, intStr);

    DrawString(dummy1, intStr, 10);

    dummy1.y := dummy1.y - verticalGap;

  end;

  dummy2.x := origin.x + horizontalGap \* (data^.Size - 1);

  dummy2.y := origin.y;

  dummy1.x := origin.x;

  dummy1.y := origin.y;

  if (cType = GridLines) or (cType = VerticalLines) then

    for i := 1 to maxVal do

    begin

      dummy1.y := dummy1.y - verticalGap;

      dummy2.y := dummy2.y - verticalGap;

      DrawLine(dummy1, dummy2, 1);

    end;

  // draw grid vertical

  dummy1.x := origin.x - 2;

  dummy1.y := origin.y;

  for i := 0 to data^.Size - 1 do

  begin

    Str(i, intStr);

    DrawString(dummy1, intStr, 10);

    dummy1.x := dummy1.x + horizontalGap;

  end;

  dummy2.x := origin.x;

  dummy2.y := origin.y - verticalGap \* maxVal;

  dummy1.x := origin.x;

  dummy1.y := origin.y;

  if (cType = GridLines) or (cType = HorizontalLines) then

    for i := 1 to data^.Size do

    begin

      dummy1.x := dummy1.x + horizontalGap;

      dummy2.x := dummy2.x + horizontalGap;

      DrawLine(dummy1, dummy2, 1);

    end;

  // draw values

  prev.x := 0;

  prev.y := 0;

  curr.x := origin.x;

  for i := 1 to data^.Size do

  begin

    currVal := MLInteger(data^.GetAt(i))^.AsInteger;

    curr.y := origin.y - Round((currVal / (maxVal \* 100)) \* maxVal \* verticalGap);

    dummy1.x := curr.x - 5;

    dummy1.y := curr.y - 5;

    DrawFilledRectangle(dummy1, 10, 10);

    if i <> 1 then

      DrawLine(prev, curr, 1);

    if (cType = Simple) or (cType = GridLines) then

    begin

      Str(currVal, intStr);

      dummy1.x := curr.x + 5;

      dummy1.y := curr.y - 20;

      DrawString(dummy1, intStr, 10);

    end;

    prev := curr;

    curr.x := curr.x + horizontalGap;

  end;

end; (\*ChartWindowObj.DrawChart\*)

(\*=== ChartApplication ===\*)

function NewChartApplication(filename: string): ChartApplication;

var

  a: ChartApplication;

begin

  New(a, Init('MiniChart', filename));

  NewChartApplication := a;

end; (\*NewChartApplication\*)

constructor ChartApplicationObj.Init(name, filename: STRING);

begin

  inherited Init(name);

  Register('ChartApplication', 'MLApplication');

  ExtractDataFromFile(filename);

end; (\*ChartApplicationObj.Init\*)

destructor ChartApplicationObj.Done;

begin

  inherited Done;

end; (\*ChartApplicationObj.Done\*)

procedure ChartApplicationObj.OpenNewWindow;

begin

  NewChartWindow(title, data)^.Open;

end; (\*ChartApplicationObj.OpenNewWindow\*)

procedure ChartApplicationObj.BuildMenus;

begin

  (\*chart types menu:\*)

  simpleLinesCommand  := NewMenuCommand('Chart Type', 'Simple',  's');

  verticalLinesCommand := NewMenuCommand('Chart Type',  'Vertical', 'v');

  horizontalLinesCommand  := NewMenuCommand('Chart Type',  'Horizontal',  'h');

  gridLinesCommand  := NewMenuCommand('Chart Type',  'Grid',  'g');

end; (\*ChartApplicationObj.BuildMenus\*)

procedure ChartApplicationObj.ExtractDataFromFile(filename: STRING);

var

  inFile: TEXT;

  line: string;

  value: integer;

begin

  assign(inFile, filename);

  reset(inFile);

  data := NewMLVector;

  if not eof(inFile) then

    readln(inFile, title);

  while not eof(inFile) do

  begin

    readln(inFile, line);

    Val(line, value);

    data^.Add(NewMLInt(value));

  end;

end; (\*ChartApplicationObj.ExtractDataFromFile\*)

(\*=== main program ===\*)

var

  a: ChartApplication;

  filename: string;

begin (\*Chart1\*)

  write('Enter input filename: '); ReadLn(filename);

  a := NewChartApplication(filename);

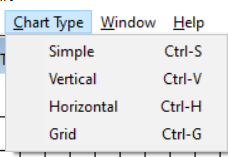
  a^.Run;

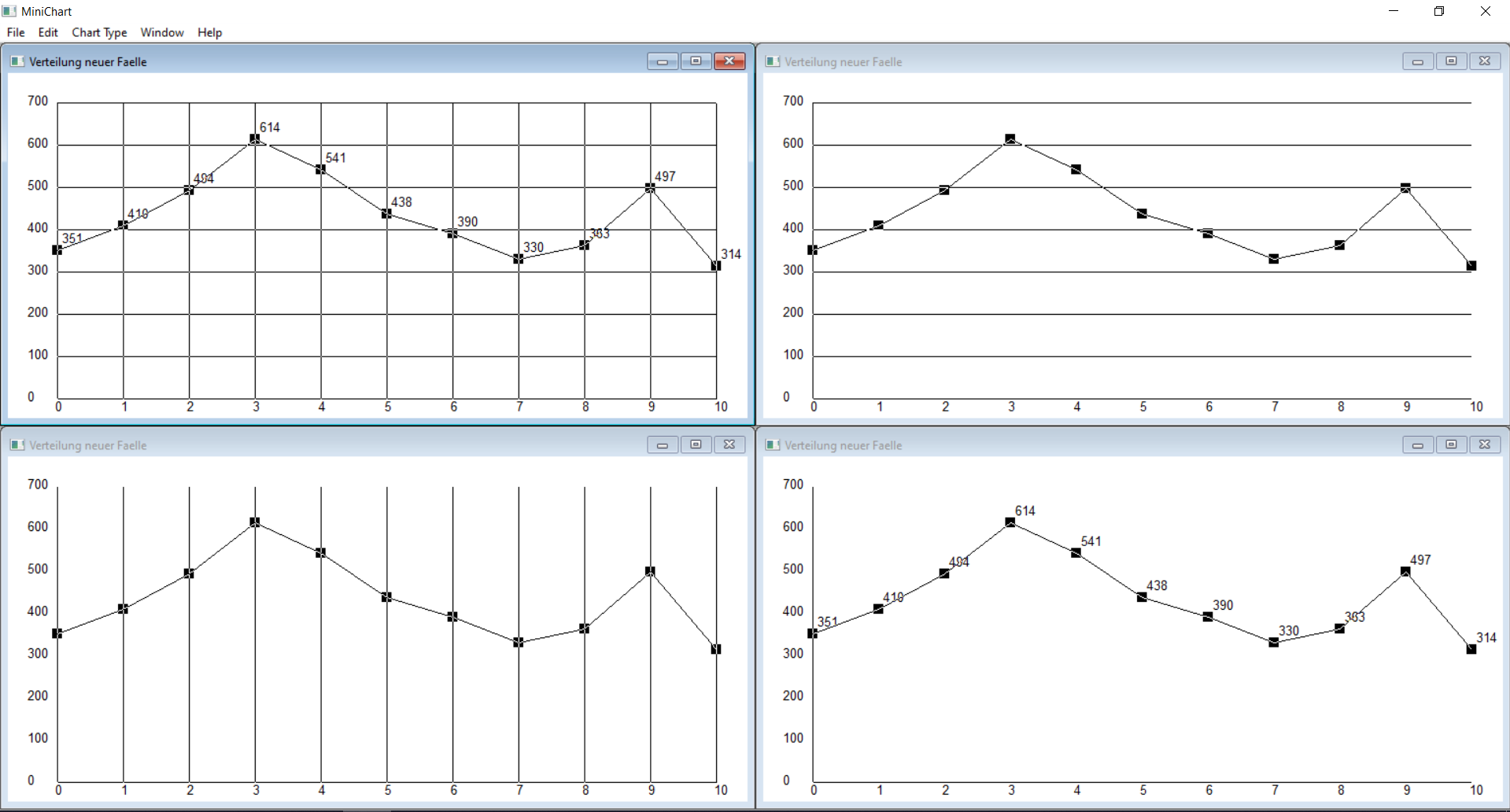
  Dispose(a, Done);

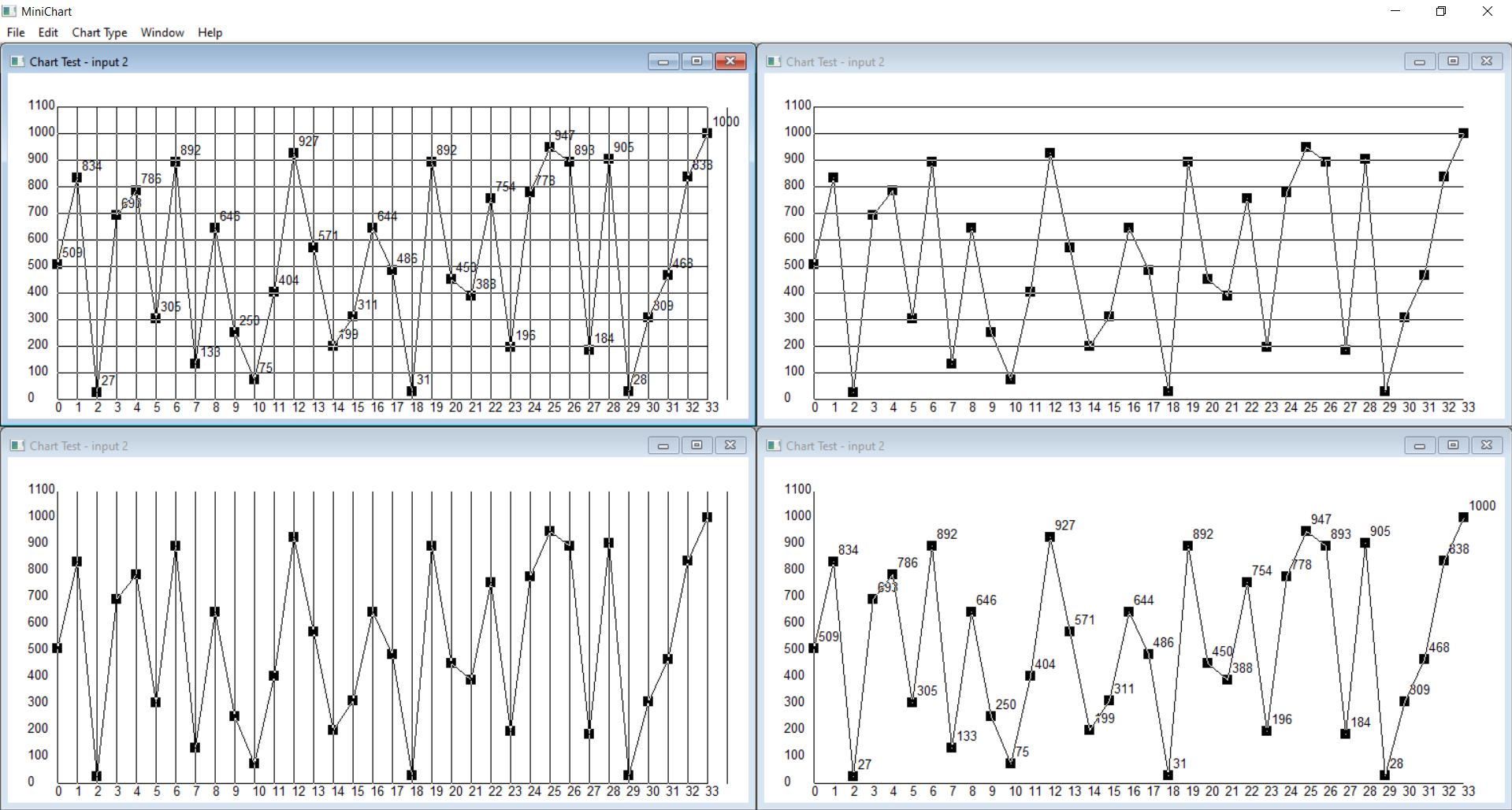
  WriteMetaInfo;

end. (\*Chart1\*)

**Tests:**







*(input files are in the zip)*