Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SoSe2005
schriftliche Prüfung im Fach Prozesstechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

In der Prozessdatenverarbeitung müssen die Ereignisse externer Geräte gespeichert werden. Entwickeln Sie daher auf Basis des 8086-Prozessors eine Toolbox zur Verwaltung externer Geräte ("Nodes") und deren Ereignisse ("Events") durch möglichst bedeutungstreue Übersetzung der Funktionen *NewNode* und *NewEvent* des Pascal-Programms *ListOfEvents*. Die Prozeduren *ListEvents* und *ListNodes*, das Hauptprogramm sowie die Ausgabe des Pascal-Programms *ListOfEvents* dienen nur zur Veranschaulichung der Aufgabenstellung.

Zur Realisierung der New-Prozedur wird die DOS-Funktion "Allocate Memory" (Funktionscode 48h) verwendet. Sie erwartet im BX-Register die Anzahl der benötigten zusammenhängenden 16-Byte Blöcke und liefert als Ergebnis im AX-Register die Segmentadresse der reservierten Blöcke (als Offsetadresse innerhalb des Segments wird Null impliziert). Der Fall "nicht genügend zur Verfügung stehender Speicher" bleibt unberücksichtigt.

Vergessen Sie nicht die hinreichende Kommentierung Ihres Assembler-Programms z.B. durch Zuordnung der Pascal-Befehle zu den Assembler-Befehlen.

Das PTL-Team wünscht viel Erfolg

```
{$X+}
Program ListOfEvents;

Type tpNode  = ^tNode;
     tpEvent = ^tEvent;
     tNode   = Record
       ID         : Word;
       firstEvent : tpEvent;
       lastEvent  : tpEvent;
       nextNode   : tpNode
     End;
     tEvent  = Record
       Class      : Word;
       nextEvent  : tpEvent
     End;

Var Nodes : tpNode;

Function NewNode(Var Nodes:tpNode;NodeID:Word):Boolean;

Var pNode : tpNode;

Begin
  If Nodes = Nil Then Begin
    New(pNode);
    pNode^.ID := NodeID;
    pNode^.firstEvent := Nil;
    pNode^.lastEvent := Nil;
    pNode^.nextNode := Nil;
    Nodes := pNode
  End Else If NodeID < Nodes^.ID Then Begin
    New(pNode);
    pNode^ := Nodes^;
    Nodes^.ID := NodeID;
    Nodes^.firstEvent := Nil;
    Nodes^.lastEvent := Nil;
    Nodes^.nextNode := pNode
  End Else If NodeID = Nodes^.ID Then
    NewNode := False
  Else
    NewNode := NewNode(Nodes^.nextNode,NodeID)
End;

Function NewEvent(Nodes:tpNode;NodeID:Word;EventClass:Word):Boolean;

Var pEvent : tpEvent;

Begin
  If Nodes = Nil Then
    NewEvent := False
  Else If NodeID < Nodes^.ID Then
    NewEvent := False
  Else If NodeID = Nodes^.ID Then Begin
    New(pEvent);
    pEvent^.Class := EventClass;
    pEvent^.nextEvent := Nil;
    If Nodes^.firstEvent = Nil Then
      Nodes^.firstEvent := pEvent;
    If Nodes^.lastEvent <> Nil Then
      Nodes^.lastEvent^.nextEvent := pEvent;
    Nodes^.lastEvent := pEvent
  End Else
    NewEvent := NewEvent(Nodes^.nextNode,NodeID,EventClass)
End;
```

```
Procedure ListEvents(Events:tpEvent);

Begin
  If Events <> Nil Then Begin
    Write(Events^.Class:6);
    ListEvents(Events^.nextEvent)
  End
End;

Procedure ListNodes(Nodes:tpNode);

Begin
  If Nodes <> Nil Then Begin
    Write(Nodes^.ID:5,':');
    ListEvents(Nodes^.firstEvent);
    WriteLn;
    ListNodes(Nodes^.nextNode)
  End
End;

Begin
  Nodes := Nil;
  NewNode(Nodes,2);
  NewNode(Nodes,4);
  NewNode(Nodes,6);
  NewNode(Nodes,8);
  NewEvent(Nodes,2,2000);
  NewEvent(Nodes,4,4000);
  NewEvent(Nodes,6,6000);
  NewEvent(Nodes,8,8000);
  NewEvent(Nodes,2,200);
  NewEvent(Nodes,4,400);
  NewEvent(Nodes,6,600);
  NewEvent(Nodes,8,800);
  NewEvent(Nodes,2,20);
  NewEvent(Nodes,4,40);
  NewEvent(Nodes,6,60);
  NewEvent(Nodes,8,80);
  NewNode(Nodes,1);
  NewNode(Nodes,3);
  NewNode(Nodes,5);
  NewNode(Nodes,7);
  NewNode(Nodes,9);
  NewEvent(Nodes,1,1000);
  NewEvent(Nodes,3,3000);
  NewEvent(Nodes,5,5000);
  NewEvent(Nodes,7,7000);
  NewEvent(Nodes,9,9000);
  NewEvent(Nodes,1,100);
  NewEvent(Nodes,3,300);
  NewEvent(Nodes,5,500);
  NewEvent(Nodes,7,700);
  NewEvent(Nodes,9,900);
  NewEvent(Nodes,1,10);
  NewEvent(Nodes,3,30);
  NewEvent(Nodes,5,50);
  NewEvent(Nodes,7,70);
  NewEvent(Nodes,9,90);
  ListNodes(Nodes)
End.
```

1:	1000	100	10
2:	2000	200	20
3:	3000	300	30
4:	4000	400	40
5:	5000	500	50
6:	6000	600	60
7:	7000	700	70
8:	8000	800	80
9:	9000	900	90

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten WS2005/6
schriftliche Prüfung im Fach Prozesstechnik
gewählter Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Übersetzen Sie auf der Basis des 8086-Prozessors möglichst bedeutungstreu die Prozeduren *GetVirtualScreen*, *FreeVirtualScreen*, *ClearVirtualScreen*, *ShowVirtualScreen*, *WriteChar* und *WriteText* des Pascal-Programms *blur_some_text*. Die übrigen Prozeduren und Funktionen sowie das Hauptprogramm dienen nur zur Veranschaulichung der Aufgabenstellung.

Vergessen Sie nicht die hinreichende Kommentierung Ihres Assembler-Programms z.B. durch Zuordnung der Pascal-Befehle zu den Assembler-Befehlen.

Das PTL-Team wünscht viel Erfolg

```
program blur_some_text;

{$M 16384, 65535, 65535}

{ Die Speicheranweisung unter Pascal ist wichtig,      }
{ da wir darauf achten muessen, dass genug fuer uns    }
{ bzw. fuer Dos frei bleibt - Pascal dient hier nur    }
{ als Rahmen in dem Sinne - wir machen alles selber!!! }

type  TVScreen = Array [0..63999] of byte;
      PVScreen = ^TVScreen;

const txt      : string = 'P T L';

var   fseg     : word;
      fofs     : word;

function MyKeyPressed : boolean; Assembler;

Asm
  mov ax,0100h
  int 16h
  mov ax,0
  jz  @loop1
  mov ax,1
  @loop1:
End;

function MyGetMem(size : word) : pointer; Assembler;

Asm
  mov  ax,4800h
  mov  bx,size
  mov  cl,4
  shr  bx,cl
  inc  bx
  int  21h
  xor  dx,dx
  xchg ax,dx
End;

procedure MyFreeMem(pntr : pointer); Assembler;

Asm
  les  di,pntr
  mov  ax,4900h
  int  21h
End;

Procedure GetVirtualScreen(var myscreen : pvscreen);

Begin
  myscreen := mygetmem (65535);
End;

Procedure FreeVirtualScreen(myscreen : pvscreen);

Begin
  myfreemem (myscreen);
End;

Procedure ClearVirtualScreen(myscreen : pvscreen; col : byte);
var cnt:word;

Begin
  for cnt:=0 to 63999 do
      myscreen^[cnt]:= col;
End;
```

```
Procedure ShowVirtualScreen(myscreen : pvscreen);
var cnt:word;

Begin
  for cnt:=0 to 63999 do
      mem[$0a000:cnt]:=myscreen^[cnt];
End;

procedure GetFont; Assembler;

Asm
  mov ax,1130h;
  mov bh,1;
  int 10h;
  mov fseg,es;
  mov fofs,bp;
End;

Procedure SetPaletteEntry(colorno : byte; r,g,b : byte);

Begin
  port[$3c8] := colorno;
  port[$3c9] := r;
  port[$3c9] := g;
  port[$3c9] := b;
End;

Procedure WaitRetrace;

Begin
  repeat until Port[$03da] and 8 = 0;
  repeat until Port[$03da] and 8 = 8;
End;

Procedure Blur(myscreen : pvscreen);
var n,k,i : word;
    origin: word;

Begin
  for i:= 318 to 63999-320 do
  begin
    origin:=       myscreen^[i-320];
    origin:=origin+myscreen^[i-001];
    origin:=origin+myscreen^[i+001];
    origin:=origin+myscreen^[i+320];
    origin:=origin shr 2;
    myscreen^[i]:=byte(origin);
  end;
End;

Procedure WriteChar(ch : char; myscreen : pvscreen;
                    x,y : word; col : byte);
var i,j,k : byte;
    pre   : word;
    opt   : word;

Begin
  pre := byte(ch)*8;
  for i:=0 to 7 do
    for j:=0 to 7 do
      for k:=0 to 3 do
      begin
        opt := (y+i * 3)*320+x+j * 3;
        if k>1 then inc(opt,318);
        if ((mem[fseg:fofs+pre+i] shl j) and 128)<>0 then
           myscreen^[opt+k] := col;
      end;
End;
```

```
Procedure WriteText(text : string; myscreen : pvscreen;
                    x,y : word; col : byte);
var n,k,i : word;

Begin
  for n:=1 to length(text) do
    writechar(text[n],myscreen,x+(n-1)*25,y,col);
End;

Function TextLength(text:string):word;

Begin
  TextLength:=length(text)*25;
End;

var vscreen : pvscreen;
    origin  : word;
    x_pos   : integer;
    y_pos   : integer;
    x_inc   : integer;
    y_inc   : integer;

begin
  x_pos:=20;
  y_pos:=30;
  x_inc:=+2;
  y_inc:=-1;
  asm
    mov ax,13h
    int 10h
  end;
  getfont;
  getvirtualscreen(vscreen);
  for origin:=000 to 063 do
    setpaletteentry(origin,origin,origin shr 1,origin shr 2);
  for origin:=064 to 127 do
    setpaletteentry(origin,127-origin,origin shr 1,origin shr 2);
  setpaletteentry(128,0,0,0);
  clearvirtualscreen(vscreen,31);
  repeat
    writetext(txt,vscreen,x_pos,y_pos,128);
    waitretrace;
    showvirtualscreen(vscreen);
    blur(vscreen);
    inc(x_pos,x_inc);
    inc(y_pos,y_inc);
    if (x_pos<05) or (x_pos>315-textlength(txt)) then x_inc:=-x_inc;
    if (y_pos<05) or (y_pos>175)                 then y_inc:=-y_inc;
  until mykeypressed;
  asm
    mov  ax,02h
    int  10h
  end;
  freevirtualscreen(vscreen);
end.
```

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SoSe2006
schriftliche Prüfung im Fach Prozesstechnik
1. Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Entwickeln Sie auf der Basis des 8086-Prozessors einen Prototypen zur simplen Textanalyse (absolute Buchstabenhäufigkeit) durch möglichst bedeutungstreue Übersetzung des Pascal-Programms *CountChars*.

Zur Realisierung der New-Prozedur und der Dispose-Prozedur werden die DOS-Funktionen "Allocate Memory" (Funktionscode 48h) und "Free Memory" (Funktionscode 49h) verwendet. Die Funktion zum Reservieren von Speicherplatz erwartet im BX-Register die Anzahl der benötigten zusammenhängenden 16-Byte Blöcke und liefert als Ergebnis im AX-Register die Segmentadresse der reservierten Blöcke (als Offsetadresse innerhalb des Segments wird Null impliziert). Der Fall "nicht genügend zur Verfügung stehender Speicher" bleibt unberücksichtigt. Die Funktion zum Freigeben von Speicherplatz erwartet im ES-Register die Segmentadresse der freizugebenden Blöcke, also eine Adresse die von der Funktion "Allocate Memory" zurückgeliefert wurde.

Vergessen Sie nicht die hinreichende Kommentierung Ihres Assembler-Programms z.B. durch Zuordnung der Pascal-Befehle zu den Assembler-Befehlen.

Das PTL-Team wünscht viel Erfolg

```
Program CountChars;

Type tpDigits = ^tDigits;
     tDigits  = Record
                  Next  : tpDigits;
                  Digit : Char
                End;

Const Digits : tpDigits = Nil;

Var S : String;

Procedure InitDigits;

Var B : Byte;
    C : Char;
    P : tpDigits;

Begin
  C := '9';
  For B := 1 To 10 Do Begin
    New(P);
    P^.Next := Digits;
    P^.Digit := C;
    Digits := P;
    C := Pred(C)
  End
End;

Procedure DestroyDigits;

Var P : tpDigits;

Begin
  While Digits <> Nil Do Begin
    P := Digits;
    Digits := Digits^.Next;
    Dispose(P)
  End
End;

Function ReadChar:Char; Assembler;

Asm
  MOV AH,8
  INT 21h
End;

Procedure ReadString(Var S:String);

Var C : Char;

Begin
  S := '';
  Repeat
    C := ReadChar;
    If (C >= #32) And (C <= #126) And (S[0] < #255) Then Begin
      S := S+C;
      Write(C)
    End Else
      If (C = #8) And (S[0] > #0) Then Begin
        S[0] := Pred(S[0]);
        Write(C,' ',C)
      End
  Until C = #13
End;
```

```
Procedure WriteDigit(B:Byte);

Var P : tpDigits;

Begin
  P := Digits;
  While B > 0 Do Begin
    P := P^.Next;
    Dec(B)
  End;
  Write(P^.Digit)
End;

Procedure WriteNumber(W:Word);

Begin
  If W > 9 Then
    WriteNumber(W Div 10);
  Write(W Mod 10)
End;

Procedure WriteString(Var S:String);

Var B : Byte;

Begin
  For B := 1 To Ord(S[0]) Do
    Write(S[B])
End;

Procedure AnalyzeString(Var S:String);

Var Abc : Array[0..25] Of Word;
    B   : Byte;
    C   : Char;

Begin
  For C := 'A' To 'Z' Do
    Abc[Ord(C)-Ord('A')] := 0;
  For B := 1 To Ord(S[0]) Do Begin
    C := S[B];
    If (C >= 'A') And (C <= 'Z') Then
      Inc(Abc[Ord(C)-Ord('A')])
  End;
  For C := 'A' To 'Z' Do Begin
    Write(C,' : ');
    WriteNumber(Abc[Ord(C)-Ord('A')]);
    WriteLn
  End
End;

Begin
  InitDigits;
  ReadString(S);
  WriteLn;
  AnalyzeString(S);
  WriteString(S);
  WriteLn;
  DestroyDigits
End.
```

Physikalisch-Technische Lehranstalt Wedel

Staatsexamen für Technische Assistenten SoSe2006
schriftliche Prüfung im Fach Prozesstechnik
2. Vorschlag

Dauer : 180 Minuten **keine externen Hilfsmittel**

Aufgabe :

Übersetzen Sie auf der Basis des 8086-Prozessors möglichst bedeutungstreu die Prozeduren *GetVirtualScreen*, *FreeVirtualScreen*, *ClearVirtualScreen* und *ShowVirtualScreen* sowie das Hauptprogramm (inklusive der zwei typisierten Konstanten und vier Variablen) des Pascal-Programms *draw_voxel*. Die übrigen Prozeduren und Funktionen dienen nur zur Veranschaulichung der Aufgabenstellung.

Vergessen Sie nicht die hinreichende Kommentierung Ihres Assembler-Programms z.B. durch Zuordnung der Pascal-Befehle zu den Assembler-Befehlen.

Das PTL-Team wünscht viel Erfolg

```
program draw_voxel;

{$M 16384, 65535, 65535}

{ Die Speicheranweisung unter Pascal ist wichtig,      }
{ da wir darauf achten muessen, dass genug fuer uns    }
{ bzw. fuer Dos frei bleibt - Pascal dient hier nur    }
{ als Rahmen in dem Sinne - wir machen alles selber!!! }

Type  TVScreen = Array [0..63999] of Byte;

      PVScreen = ^TVScreen;

      { hier zwei Datentypen um spaeter ueber den Pointer   }
      { ein Array zu indizieren welches sich nach Dimension }
      { eines virtuellen Grafik Bildspeichers ansprechen    }
      { lassen soll...                                      }

Const Scal_Fact : Array[0..6,0..3] of Word =

      ( ( 2, 0,   158, 0  )   ,
        ( 2, 286, 136, 10 )   ,
        ( 2, 667, 106, 20 )   ,
        ( 3, 200, 97,  30 )   ,
        ( 4, 0,   79,  40 )   ,
        ( 5, 333, 59,  50 )   ,
        ( 8, 0,   39,  60 ) ) ;

      { Dieses konstante Array bestimmt die Skalierung sowie }
      { Offset Position innerhalb der Quell Grafik Datei fuer}
      { fuer die einzelnen zu malenden Rechtecke... Aufbau : }
      { breite - nachkommastelle - anzahl pixel - offset inc }

Const Filename : String =

      'voxel.raw' + #0;

      { Dateiname der zu ladenden Grafik Pixel Map, die PixelMap  }
      { ist so aufgebaut, dass sie aus 160 Pixel der Breite sowie }
      { aus 400 Zeilen besteht. Ein Pixel repraesentiert gemaess  }
      { seiner Wertigkeit die Hoehe eines daraus entstehenden     }
      { Rechtecks...                                              }

Function MyKeyPressed : boolean; Assembler;

      { Ueberprueft ob eine Taste gedrueckt wurde und gibt }
      { entsprechend einen boolean im ah Register zurueck  }

Asm
  mov ax,0100h
  int 16h
  mov ax,0
  jz  @loop1
  mov ax,1
@loop1:
End;

Function GetMyKey : word; Assembler;

     { Liest eine Taste aus und gibt sie ueber AX zurueck   }
     { Funktionstasten stehen im AH Register und "normale"  }
     { kommen als Rueckgabe in den unteren 8 Bits zurueck   }

Asm
  xor  ax,ax
  int  16h
End;
```

```
Procedure ReadLandScape(pntr:PVScreen); Assembler;

     { Auslesen einer gegebenen Datei in einen }
     { reservierten Speicherbereich            }

Asm
  push ds
  mov  ax,3d02h
  lea  dx,filename +1
  int  21h
  mov  bx,ax
     { ^^ Datei ueber den Namen oeffnen und ein Filehandle }
     { in dem BX Register festhalten...                    }
  mov  ax,4200h
  xor  cx,cx
  xor  dx,dx
  int  21h
    { ^^ Filehandle innerhalb Datei ueber DWORD cx sowie dx }
    { auf den Anfang positionieren...                       }
  mov  ax,3f00h
  mov  cx,64000
  lds  dx,pntr
  int  21h
    { ^^ genau 64000 Bytes in den Speicherbereich ds:dx einlesen }
  mov  ax,3e00h
  int  21h
    { ^^ Datei wieder schliessen... }
  pop  ds
End;

Function MyGetMem(size : word) : pointer; Assembler;

    { Speicherbereich reservieren; es wird ein Speicherbereich     }
    { dessen Groesse im BX Register erwartet wird, wobei diese auf }
    { Paragraphen zu jeweils 16 Bytes runtergerechnet werden,      }
    { reserviert und der Pointer auf diesen Speicherbereich via    }
    { AX:DX zurueck gegeben                                        }

Asm
  mov  ax,4800h
  mov  bx,size
  mov  cl,4
  shr  bx,cl
  inc  bx
  int  21h
  xor  dx,dx
  xchg ax,dx
End;

Procedure MyFreeMem(pntr : pointer); Assembler;

    { Einen schon reservierten Speicherbereich ueber ES:DI wieder }
    { freigeben                                                   }

Asm
  les  di,pntr
  mov  ax,4900h
  int  21h
End;

Procedure GetVirtualScreen(var myscreen : pvscreen);

Begin
  myscreen := mygetmem (65535);
End;
```

```
Procedure FreeVirtualScreen(myscreen : pvscreen);

Begin
  myfreemem (myscreen);
End;

Procedure ClearVirtualScreen(myscreen : pvscreen; col : byte);

var cnt   :word;

Begin
  for cnt:=0 to 63999 do
      myscreen^[cnt]:= col;
End;

Procedure ShowVirtualScreen(myscreen : pvscreen);

var cnt   :word;

Begin
  for cnt:=0 to 63999 do
      mem[$0a000:cnt]:=myscreen^[cnt];
End;

Procedure SetPaletteEntry(colorno : byte; r,g,b : byte);

Begin
  port[$3c8] := colorno;
    { Port zum Bestimmen des Farbtabellenindex          }
  port[$3c9] := r;
    { Rot Wertigkeit in der Grafikkarte programmieren   }
  port[$3c9] := g;
    { Gruen Wertigkeit in der Grafikkarte programmieren }
  port[$3c9] := b;
    { Blau Wertigkeit in der Grafikkarte programmieren  }
End;

Procedure WaitRetrace;

Begin
  repeat until Port[$03da] and 8 = 0;
  { Zuerst ueberpruefen, ob gerade ein Waitretrace ablaeuft, }
  { um diesen dann solange abzuwarten                        }
  repeat until Port[$03da] and 8 = 8;
  { auf einen neuen Waitretrace / Strahlenruecklauf warten }
End;

Procedure DrawRectangle(x,y,width,height:word;color:byte;
                        myscreen:pvscreen);

var cnt1  :word;
    cnt2  :word;

begin
  for cnt1:= 1 to height do
    for cnt2:= 1 to width do
        myscreen^[(y+cnt1-1)*320+(x+cnt2-1)]:=color;
end;
```

```
Procedure DrawVoxelPlane(src,dst:pvscreen;index:integer);

var cnt1  :word;      { cnt1 steht fuer die einzelnen Ebenen die uebereinan-
                        der gemalt werden }
    cnt2  :word;
                      { cnt2 steht fuer die Anzahl zu zeichenden Rechtecke }
    cnt3  :word;
                      { cnt3 entspricht dem Offset auf der horizontalen
                        Ebene des Grafik Bildschirms }
    offs  :word;      { Offset entspricht der Position innerhalb der Quell
                        Grafik errechnet aus y Pos + xpos + centeroffset }
    inc1  :word;      { inc1 berechnet den Nachkommaanteil im Bereich 1/1000
                        fuer eine angepasste Breitendarstellung }
    inc2  :boolean;   { inc2 ergibt sich aus einem Ueberlauf einer fortlau-
                        fenden Addition von inc1 }
    width :word;      { eigentliche Breite eines zu malenden Rechteckes }
    origin:word;      { Farbwert aus der Quell Grafik }

begin
  { der Algorithmus laeuft so ab, als das 6 einzelne Ebenen uebereinander
    gezeichnet werden, wobei von der hintersten }
  { Ebene angefangen mit den meisten Bildinformationen in der Horizontalen
    bis zur vordersten Ebene gezeichnet wird. }
  { Innerhalb der aeussersten Schleife wird zuerst fuer OFFS die Position
    innerhalb der Quell Grafik Bitmap, welche }
  { die Hoeheninformation fuer die Landschaft enthaelt, der Offset berech-
    net sowie innerhalb der INC Variablen ein Nach- }
  { kommaanteil fuer die Breite eines Rechteckes, welches ein Hoehenpixel
    aus der Quell Datei repraesentiert, festge- }
  { halten. Innerhalb der zweiten Schleife wird darauffolgend fuer jede
    Ebene gemaess Anzahl aus dem Konstantenarray ein }
  { Rechteck ueber die in cnt3 errechnete X Position gezeichnet. Darauf
    folgt die Nachkommastellenberechnung und das }
  { "Geheimnis" sollte hiermit entlueftet sein. Zum Ende der aeusseren
    Schleife wird noch einmal auf ein Ueberlauf }
  { innerhalb der Quellgrafik acht gegeben... }
  for cnt1:=0 to 6 do
  begin
    offs:=index*160;
    offs:=offs+scal_fact[cnt1,3];
    cnt3:=0;
    inc1:=scal_fact[cnt1,1];
    inc2:=false;
    for cnt2:=0 to scal_fact[cnt1,2] do
    begin
      origin:=src^[offs+cnt2];
      width:=scal_fact[cnt1,0];
      if inc2 then inc(width);
      drawrectangle(cnt3,199-origin shr 1 div (cnt1+1),width,
                    origin shr 1 div (cnt1+1),origin,dst);
      cnt3:=cnt3+scal_fact[cnt1,0];
      if inc2 then
      begin
        inc2:=false;
        inc(cnt3);
      end;
      inc1:=inc1+scal_fact[cnt1,1];
      if inc1>=1000 then
      begin
        inc1:=scal_fact[cnt1,1];
        inc2:=true;
      end;
    end;
    dec (index);
    if index<0 then index:=399;
  end;
end;
```

```
var landscape : pvscreen;
    vscreen   : pvscreen;
    origin    : integer;
    mykey     : word;

begin
  asm
    mov ax,13h
    int 10h
  end;
  { Grafik Modus einschalten }
  getvirtualscreen(vscreen);
  { virtuellen Grafik Speicher reservieren zum vorbereiten der
    Ausgabegrafik }
  getvirtualscreen(landscape);
  { Speicherbereich fuer die Quellinformationen anfordern }
  for origin:=000 to 127 do
      setpaletteentry(origin,origin shr 1,origin shr 2,origin shr 3);
  for origin:=128 to 255 do
      setpaletteentry(origin,(255-origin) shr 1,origin shr 2,origin shr 3);
  { Eine Farbpalette von braun nach gruen ueber 256 Eintraegen aufbauen }
  clearvirtualscreen(vscreen,0);
  { Virtuellen Grafikschirm loeschen }
  readlandscape(landscape);
  { Einlesen der Landschaft }
  origin:=0;
  repeat
    drawvoxelplane(landscape,vscreen,origin);
    { einen Bildschirm berechnen }
    waitretrace;
    { Strahlenruecklauf abwarten }
    showvirtualscreen(vscreen);
    { virtuellen Bildschirm uebertragen }
    clearvirtualscreen(vscreen,0);
    { virtuellen loeschen }
    if mykeypressed then mykey:=getmykey;
    { eventuell Taste auslesen }
    if mykey shr 8 = 72 then
    begin
      inc (origin);
      if origin=400 then origin:=0;
      mykey:=0;
    end;
    { Bewegung der Cursor Tasten als Position innerhalb der Quell Bitmap
      festhalten }
    if mykey shr 8 = 80 then
    begin
      dec (origin);
      if origin<0 then origin:=399;
      mykey:=0;
    end;
  until (mykey and $7F=27);
  { bis ESC gedrueckt wurde }
  asm
    mov  ax,02h
    int  10h
  end;
  { Textmodus aktivieren }
  freevirtualscreen(vscreen);
  freevirtualscreen(landscape);
end.
```