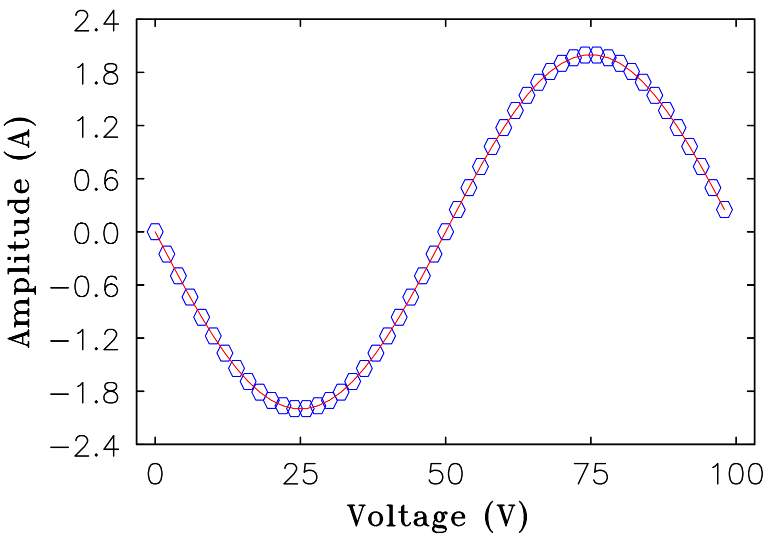
Benutzeranleitung für die WinMagra

Wolfgang Schöpf und Bert Reimann

(Experimentalphysik V, Universität Bayreuth, D-95440 Bayreuth, Germany)

31. März 2004



Inhaltsverzeichnis

Kapitel 1

Installation und Verzeichnisse

1.1 Installation der WinMagra–Distribution

WICHTIG: Die WinMagra erwartet, dass sie sich in "C:\grafik\winmagra" befindet!

Für eine Neu–Installation der WinMagra sind folgende Schritte durchzuführen:

- Erstellen eines Verzeichnisses "C:\grafik".
- Herunterladen der WinMagra als Datei "install.exe" (selbstentpackende Datei) nach "C:\grafik". (Dies ist momentan nur innerhalb der Experimentalphysik V möglich! Externe Nutzer senden bitte eine e-mail an "wolfgang.schoepf@uni-bayreuth.de".)
- "install.exe" ausführen; dabei werden erstellt:
  - install.bat
  - pkzip25.exe
  - winmagra.zip
- "install.bat" ausführen; dadurch werden alle Verzeichnisse und Dateien erstellt.
- Die Watcom–Library wird nur erstellt, wenn sich ein Watcom-Compiler auf dem Rechner befindet.

Falls auf dem Rechner bereits eine WinMagra–Installation existiert, kann die neue Distribution gemäß den oben angegebenen Schritten einfach über die alte drüber installiert werden. Liebhaber 'sauberer' Installationen ohne Ballast können natürlich auch zunächst die alte Distribution entfernen (Löschen des Verzeichnisses "C:\grafik\winmagra" samt aller Unterverzeichnisse). Das Ergebnis ist in Bezug auf die Funktionalität das gleiche. Ein spezielles Update–Programm der WinMagra ist nicht vorgesehen.
Beim Erstellen von Programmen ist noch Folgendes zu beachten:

- Das Verzeichnis "C:\grafik\winmagra\include" muss in den include–Pfad des jeweiligen Compilers aufgenommen werden.
- Das Verzeichnis "C:\grafik\winmagra\dll" muss in die Umgebungsvariable "PATH" aufgenommen werden.
- Bei Verwendung des "Microsoft Visual C/C++"–Compilers muss bei Erstellen der exe–Datei die Bibliothek "C:\grafik\winmagra\dll\winmagra.lib" dazugelinkt werden (die mitgelieferte Batch–Datei "vcpp.bat" macht dies automatisch und setzt auch den include–Pfad richtig).
- Bei Verwendung des Watcom–Compilers muss bei Erstellen der exe–Datei die Bibliothek "C:\grafik\winmagra\lib\winmagra.lib" dazugelinkt werden (die mitgelieferte Batch–Datei "make95.bat" macht dies automatisch und setzt auch den include–Pfad richtig).

- Bei Kommandozeilen–Aufruf des jeweiligen Compilers durch Verwendung der mitgelieferten Batch-Datei “make95.bat” bzw. “vcpp.bat” muss diese gegebenenfalls angepasst werden, um eigene include-Dateien und Bibliotheken einzubinden.

WICHTIG: Die WinMagra erwartet, dass sie sich in “C:\grafik\winmagra” befindet!

1.2 Verzeichnisse

Nach einer Installation gemäß den in Abschnitt 1.1 angegebenen Schritten wird folgender Verzeichnisbaum erstellt:

```
 Verzeichnis von C:\grafik\winmagra

01.05.2003  10:18       <DIR>          .
01.05.2003  10:18       <DIR>          ..
30.04.2003  15:48       <DIR>          beispiel
30.04.2003  15:48       <DIR>          bin
30.04.2003  15:48       <DIR>          dll
30.04.2003  15:48       <DIR>          include
30.04.2003  15:48       <DIR>          lib
30.04.2003  15:48       <DIR>          rc
30.04.2003  15:48       <DIR>          src
               0 Datei(en)              0 Bytes
```

Die einzelnen Unterverzeichnisse beinhalten folgende Dateien:

\beispiel:	Beispielprogramme zur Benutzung der WinMagra
\bin:	Kommandozeilen–Steuerung des Watcom–Compilers (“make95.bat”) bzw. des “Microsoft Visual C/C++”–Compilers (“vcpp.bat”)
\dll:	dynamische Bibliotheken, sowie die “winmagra.lib” für Microsoft Visual C/C++
\include:	sämtliche header–Dateien
\lib:	WinMagra–Fonts “magfont.bin”, sowie die “winmagra.lib” für den Watcom–Compiler
\rc:	WinMagra–Resourcen (Icons, Cursor, Lupe, etc.)
\src:	sämtlich Quellcodes der WinMagra

Zum Erzeugen einer WinMagra–Anwendung unbedingt erforderlich sind die Verzeichnisse **\dll**, **\include** und **\lib**, bei Verwendung der Kommandozeilen–Steuerung (“make95.bat” für den Watcom–Compiler bzw. “vcpp.bat” für den “Microsoft Visual C/C++”–Compiler) auch noch **\bin**.

1.3 Libraries

1.3.1 Microsoft Visual C/C++ Compiler

Bei Benutzung des “Microsoft Visual C/C++”–Compilers wird die WinMagra mit einer dynamischen Bibliothek (dll) verwendet. Hierzu wird bei Erstellen der exe–Datei die Bibliothek “C:\grafik\winmagra\dll\winmagra.lib” zum Programm gelinkt. Zur Laufzeit wird dann nach den Dateien “winmagra.dll”, “winmagrares.dll”, “wndclass.dll”, “base.dll” und “values.dll” gesucht, welche sich im Verzeichnis “C:\grafik\winmagra\dll” befinden. Deshalb muss dieses Verzeichnis in den Suchpfad für das Betriebssystem aufgenommen werden.

Dieses Vorgehen hat den Vorteil, dass nach einer Neuinstallation der WinMagra (z.B. nach Behebung eines Fehlers) Anwenderprogramme nicht neu erstellt werden müssen, sondern zur Laufzeit automatisch die neue “winmagra.dll” und damit der vom Fehler befreite Code eingebunden wird.

Falls der Compiler von der Kommandozeile aus gesteuert werden soll, muss die mitgelieferte Batch-Datei “vcpp.bat” gegebenenfalls angepasst werden, um eigene include-Dateien und Bibliotheken einzubinden.

1.3.2 Watcom–Compiler

Bei Benutzung des Watcom–Compilers wird die WinMagra mit einer statischen Bibliothek (lib) verwendet. Hierzu wird bei Erstellen der exe–Datei die Bibliothek "C:\grafik\winmagra\lib\winmagra.lib" zum Programm gelinkt. Zur Laufzeit wird zum Einbinden der Resourcen nach der Datei "winmagrares.dll" gesucht, welche sich im Verzeichnis "C:\grafik\winmagra\dll" befindet. Deshalb muss auch hier dieses Verzeichnis in den Suchpfad für das Betriebssystem aufgenommen werden.

Nach einer Neuinstallation der WinMagra müssen also bei Verwendung des Watcom–Compilers Anwenderprogramme neu gelinkt werden, damit die neue "winmagra.lib" eingebunden wird.

Falls der Compiler von der Kommandozeile aus gesteuert werden soll, muss die mitgelieferte Batch-Datei "make95.bat" gegebenenfalls angepasst werden, um eigene include-Dateien und Bibliotheken einzubinden.

Kapitel 2

Beispiele zur Benutzung

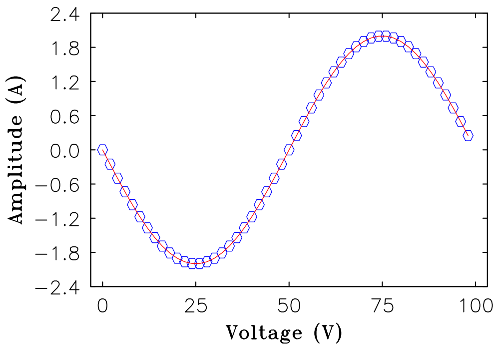
2.1 Beispiel 1: Eine Box mit Achsen, Linien und Symbolen

Das erste Beispiel zeigt, wie eine Arbeitsfläche und eine Box auf dieser Arbeitsfläche geöffnet werden. Die Achsen werden dimensioniert und beschriftet, und es werden Datenpunkte als Symbole und als Linie ins Diagramm eingezeichnet.
Das Programm hat den Namen "beispiel1.cpp" und liegt im Verzeichnis "C:\grafik\winmagra\beispiel".

```
// Programm "beispiel1.cpp"
#include "winmagra.h"

void main()
{
// hier erzeugen wir uns nur ein paar Daten
float x[50],y[50];
    for(int i=0;i<50;i++) {
        x[i]=2*i;
        y[i]=2.*sin(2.*3.14159*i/50.+3.14159);
    }
// der eigentliche Plot beginnt hier
  // wir brauchen eine Arbeitsflaeche vom Typ 'desk' und
  // eine Box vom Typ 'boxclass' auf dieser Arbeitsflaeche
desk dvi;
boxclass box(dvi);
      // Auswahl des Ausgabegeraetes mit Namen "test1"; hier: Bildschirm und Postscript-File
      // der Desk wird an die Bildschirm-Koordinaten (200,300) plaziert
    dvi.open(SCREEN|EPS_FILE,"test1","pos=(200,300)");
      // die Box wird auf der Arbeitsflaeche positioniert und initalisiert,
      // mit einem Koordinatensystem versehen, und an den Achsen beschriftet
    box.init(30,30,110,85,0.97);
    box.dim(0,100,-2.4,2.4,25,.6);
    box.label("Voltage (V) ","Amplitude (A)");
      // die Daten werden als Symbole gezeichnet und durch eine Linie verbunden
    box.showpoints(x,y,50,_BLUE,HEXAGON,1.5);
    box.connectpoints(x,y,50,_RED,1);
}
```

Die Ausgabe des Programms sieht so aus:



2.2 Beispiel 2: Zwei Boxen übereinander zeichnen

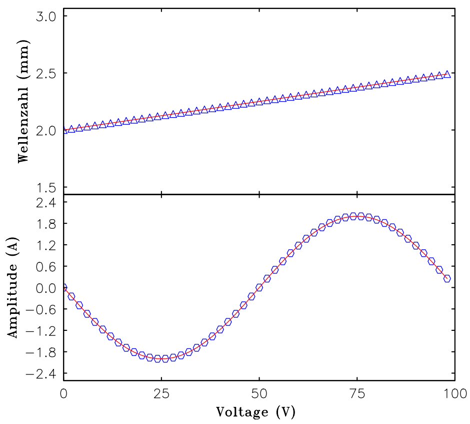
Dieses Beispiel zeigt, wie auf einer Arbeitsfläche zwei Boxen, die eine gemeinsame x–Achse besitzen, übereinander gezeichnet werden können. Dies kann natürlich beliebig erweitert werden (3 Boxen, 4 Boxen, etc.).

Das Programm hat den Namen "beispiel2.cpp" und liegt im Verzeichnis "C:\grafik\winmagra\beispiel".

```
// Programm "beispiel2.cpp"
#include "winmagra.h"

void main()
{
// hier erzeugen wir uns nur ein paar Daten
float x[50],y[50],z[50];
    for(int i=0;i<50;i++) {
        x[i]=2*i;
        y[i]=2.*sin(2.*3.14159*i/50.+3.14159);
        z[i]=i/100.+2.;
    }
// der eigentliche Plot beginnt hier
  // wir brauchen eine Arbeitsflaeche vom Typ 'desk' und
  // zwei Boxen vom Typ 'boxclass' auf dieser Arbeitsflaeche
desk dvi;
boxclass box1(dvi);
boxclass box2(dvi);
      // Auswahl des Ausgabegeraetes mit Namen "test2"; hier: Bildschirm und Postscript File
    dvi.open(SCREEN|EPS_FILE,"test2");
      // die Boxen werden auf der Arbeitsflaeche positioniert und initalisiert,
      // mit einer gemeinsamen x-Achse versehen und beschriftet
    box1.init(25,15,105,53,1,0.96);
    box2.init(25,53,105,91,1,0.96);
    box1.dim(0,100,-2.4,2.4,25,.6);
    box2.dim(0,100,1.5,3.,25,.5);
    box1.label("Voltage (V)","Amplitude (A)");
    box2.label("","Wellenzahl (mm)");
      // die Daten werden als Symbole gezeichnet und durch eine Linie verbunden
    box1.showpoints(x,y,50,_BLUE,HEXAGON,1.5);
    box1.connectpoints(x,y,50,_RED,1);
    box2.showpoints(x,z,50,_BLUE,TRIANGLE,1.5);
    box2.connectpoints(x,z,50,_RED,1);
}
```

Die Ausgabe des Programms sieht so aus:



2.3 Beispiel 3: Ein Bitmap darstellen

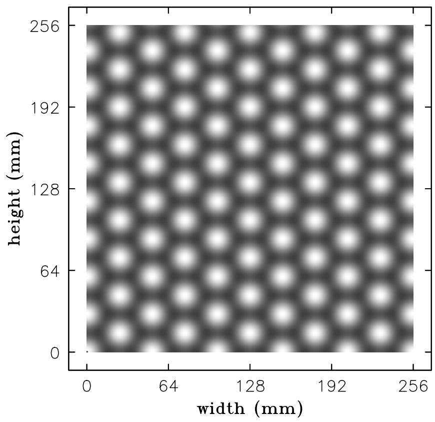
In diesem Beispiel werden zwei Möglichkeiten gezeigt, ein Bitmap in einer Box darzustellen. Zum einen kann das Bitmap mit Hilfe der boxclass–Funktion "showbmp(. . .)" direkt gezeigt werden (beispiel3a.cpp), zum anderen kann es mit Hilfe der bitmap–Klasse und der Funktion "load(. . .)" eingelesen und dann gezeigt werden (beispiel3b.cpp). Der Vorteil der zweiten Variante ist, dass das Bitmap dann innerhalb des Programms auch manipuliert werden kann. Mit Hilfe der bitmap–Funktion “save(...)” kann ein Bitmap gespeichert werden. Die Bitmap–Klasse kann also unabhängig von einer Grafik–Darstellung dazu dienen, ein beliebiges Graustufenformat in ein Bitmap umzuwandeln.

Die beiden Beispielprogramme heißen “beispiel3a.cpp” und “beispiel3b.cpp” und liegen im Verzeichnis “C:\grafik\winmagra\beispiel”.

```
// Programm "beispiel3a.cpp"
#include "winmagra.h"

void main()
{
  // wir brauchen eine Arbeitsflaeche vom Typ 'desk' und
  // eine Box vom Typ 'boxclass' auf dieser Arbeitsflaeche
desk surf;
boxclass bitbox(surf);
      // Auswahl des Ausgabegeraetes mit Namen "test3a"; hier: Bildschirm und Postscript File
    surf.open(SCREEN|EPS_FILE,"test3a");
      // die Box wird auf der Arbeitsflaeche positioniert und initalisiert,
      // mit einem Koordinatensystem versehen, und an den Achsen beschriftet
    bitbox.init(20,20,90,90,0.95,0.95);
    bitbox.dim(0,256,0,256,64,64);
    bitbox.label("width (mm)","height (mm)",.5);
      // das File "bild.bmp" wird eingelesen, und das Bitmap in der Box dargestellt
    bitbox.showbmp(0,0,256,256,"bild");
}
```

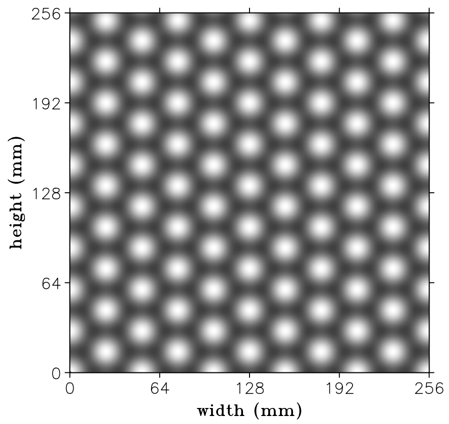
Die Ausgabe des Programms sieht so aus:




```
// Programm "beispiel3b.cpp"
#include "winmagra.h"

void main()
{
  // wir brauchen ein Objekt vom Typ 'bitmap' mit der richtigen Groesse zum Aufnehmen des Bitmaps
  // dieser Aufruf muss vor dem Aufruf der Grafikobjekte stattfinden!!
bitmap bmp(256,256);
  // wir brauchen eine Arbeitsflaeche vom Typ 'desk' und
  // eine Box vom Typ 'boxclass' auf dieser Arbeitsflaeche
desk surf;
boxclass bitbox(surf);
      // Auswahl des Ausgabegeraetes mit Namen "test3b"; hier: Bildschirm und Postscript File
    surf.open(SCREEN|EPS_FILE,"test3b");
      // die Box wird auf der Arbeitsflaeche positioniert und initalisiert (hier: Ticks
      // nach aussen), mit einem Koordinatensystem versehen, und an den Achsen beschriftet
    bitbox.init(20,20,90,90);
    bitbox.dim(0,256,0,256,64,64,1,boxclass::OUTTICKS);
    bitbox.label("width (mm)","height (mm)",.5);
      // das File "bild.bmp" wird geladen, und das Bitmap auf der Arbeitsflaeche dargestellt
      // hier muessen desk-Koordinaten verwendet werden, da putbmp eine Funktion der desk-Klasse ist!
    bmp.load("bild");
    surf.putbmp(20,20,90,90,bmp);
}
```

Die Ausgabe des Programms sieht so aus:



2.4 Beispiel 4: Einlesen von Daten mit den "GetData"–Klassen

Dieses Beispiel zeigt, wie mit Hilfe der "GetData"–Klassen Daten aus einem File eingelesen und dargestellt werden können. Es stehen Klassen für integer–Daten ("GetIntData"), float–Daten ("GetFloatData") und double–Daten ("GetDoubleData") zur Verfügung. Diese Klassen können natürlich auch unabhängig von irgendwelchen Grafiken zum Einlesen von Daten verwendet werden.

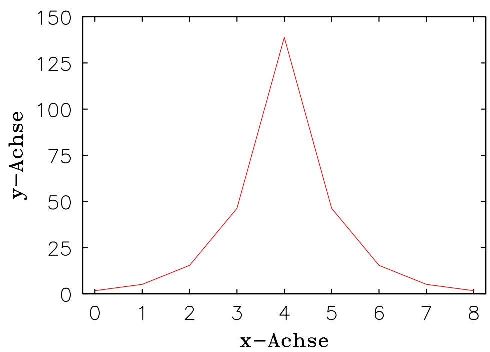
Das Programm hat den Namen "beispiel4.cpp" und liegt im Verzeichnis "C:\grafik\winmagra\beispiel".

```
// Programm "beispiel4.cpp"
#include "winmagra.h"

void main()
{
  // zum Aufnehmen der Daten fuer x- und y-Achse
float *X,*Y;
  // wir brauchen ein Objekt vom Typ 'GetFloatData'
GetFloatData GFD;
      // Einlesen von Spalte 1 als x-Achse und Spalte 3 als y-Achse aus
      // File "beispiel4.dat" und Zuordnen der Daten zu unseren Arrays
    GFD("beispiel4.dat","u 1:3");
    X=GFD(GetFloatData::XACHSE);
    Y=GFD(GetFloatData::YACHSE);

// der eigentliche Plot beginnt hier
desk dvi;
boxclass box(dvi);
    dvi.open(SCREEN|EPS_FILE,"test4");
    box.init(30,30,110,85,0.97);
      // die Koordinaten werden aus den eingelesenen Daten bestimmt,
      // die Ticks werden automatisch berechnet
    box.dim(GFD.getXMIN(),GFD.getXMAX(),GFD.getYMIN(),GFD.getYMAX(),0,0);
    box.label("x-Achse","y-Achse");
      // die Daten werden durch eine Linie verbunden gezeichnet
    box.connectpoints(X,Y,GFD.getDataLength(),_RED,1);
}
```

Die Ausgabe des Programms sieht so aus:



Kapitel 3

Farben, Symbole, Schriftarten und Koordinatensysteme

Zum Darstellen und Beschriften der Grafiken kann innerhalb der WinMagra an mehreren Stellen auf unterschiedliche Farben, Symbole und Schriftarten zurückgegriffen werden.

3.1 Farben

Die Zeichenfarbe kann von verschiedenen Funktionen der "desk"–Klasse, der "boxclass"–Klasse, der "fontclass"–Klasse und der "symbol"–Klasse gesetzt werden. Die Variable '`int co`' bzw. '`enum COLOR co`' kann dabei aus folgenden Werten ausgewählt werden:

```
co: _BLACK, _RED, _GREEN, _BLUE, _YELLOW, _MAGENTA, _VIOLET, _GREY0,
    _GREY1, _GREY2, _GREY3, _GREY4, _GREY5, _GREY6, _GREY7, _WHITE
```

Falls keine Farbe explizit angegeben wird, wird normalerweise die Standardfarbe '`_BLACK`' übernommen.

3.2 Symbole

Zum Darstellen von Datenpunkten können von verschiedenen Funktionen der "boxclass"–Klasse und der "symbol"–Klasse Symbole gesetzt werden. Die Variable '`enum SYMBOL sy`' kann dabei aus folgenden Werten ausgewählt werden:

```
sy:  CROSS, SQUARE, TRIANGLE, ROMBUS, HEXAGON, CIRCLE, _TRIANGLE, HOURGLASS
```

Falls kein Symbol explizit angegeben wird, wird normalerweise das Standardsymbol 'SQUARE' übernommen.
Die Symbole können durch ODER (|) mit einer Wanddicke

```
ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT
```

verknüpft werden. Der cast '`SYMBOL(HEXAGON|THREE)`' erzeugt also ein Hexagon mit 3 Linien Wanddicke. Bei einer Symbolgröße von '1' ergibt eine Wanddicke von '`EIGHT`' ein ausgefülltes Symbol!
Eine Ausnahme stellen die 'showerrorbars'–Funktionen der "boxclass"–Klasse dar. Hier sind als Symbole nur die Angaben für Fehlerbalken in x– und y–Richtung sinnvoll:

```
sy:  XBARS, YBARS
```

Auch bleibt die Wanddicke fest auf '`ONE`' eingestellt.

3.3 Schriftarten

Zum Beschriften von Grafiken können von verschiedenen Funktionen der "boxclass"–Klasse und der "fontclass"–Klasse Schriftarten gesetzt werden. Die Variable '`int fo`' kann dabei aus folgenden Werten ausgewählt werden:

```
fo:  ARIAL, ROMAN, GREEK
```

Falls keine Schriftart explizit angegeben wird, wird normalerweise der Standardfont '`ROMAN`' übernommen.

Die jeweilige Schriftart kann durch ODER (|) mit einer Wanddicke

```
SMALL, BOLD
```

verknüpft werden. Jeder Font existiert also in drei Wanddicken: normal, dünn und dick.

Einzelne Textabschnitte können durch '`_{abc}`' oder '`^{abc}`' tief– oder hochgestellt werden. Durch '`$abc$`' wird der jeweilige Text im Font '`GREEK`' dargestellt. In diesem Font befinden sich z.B. auch mathematische Zeichen und verschiedene Sonderzeichen.

Eine Übersicht aller druckbarer Zeichen ist im Anhang angegeben.

3.4 Koordinatensysteme

Beim Zeichnen von Grafiken muss ganz allgemein zwischen zwei Koordinatensystemen unterschieden werden:

- das Koordinatensystem der Arbeitsfläche (hier der 'desk' aus der "desk"–Klasse)
- das benutzerdefinierte Koordinatensystem (hier die 'box' aus der "boxclass"–Klasse)

Das 'desk'–Koordinatensystem geht normalerweise von $x = 0, y = 0$ (links unten) bis $x = 133, y = 100$ (rechts oben). Bei doppelter Größe der Arbeitsfläche (Angabe des Parameters '`DOUBLE_SIZE`' in '`dvi.open()`', s. nächstes Kapitel) liegt rechts oben bei $x = 133, y = 200$.

Die Funktionen der "desk"–Klasse, der "fontclass"–Klasse und der "symbol"–Klasse verwenden alle das 'desk'–Koordinatensystem, ebenso wie die Funktion '`init(...)`' aus der "boxclass"–Klasse.

Alle anderen Funktionen der "boxclass"–Klasse verwenden das durch die "boxclass"–Funktion '`dim(...)`' festgelegte, benutzerdefinierte Koordinatensystem.

Kapitel 4

Klassen und Funktionen

4.1 Klasse “desk”

Die “desk”–Klasse stellt die Arbeitsfläche des Grafiksystems dar. Sie sorgt dafür, dass die jeweiligen Ausgabegeräte richtig angesprochen und verwaltet werden.

4.1.1 Konstruktor

```
desk(HINSTANCE hInstance=0,HINSTANCE prevhInstance=0,LPSTR lpStr=NULL,int nCmdShow=0);
```

Hiermit wird die Arbeitsfläche erstellt, auf der die verschiedenen Ausgabegeräte (`SCREEN`, `EPS_FILE`, `PLT_FILE`) aktiviert werden können. Die Parameter sind optional und werden vom Benutzer normalerweise nicht verwendet.

Beispiel: `desk dvi;`

Ein Objekt mit Namen ‘dvi’ vom Typ ‘desk’ wird erzeugt.

In einem Programm können auch mehrere Arbeitsflächen vom Typ ‘desk’ erstellt werden. Für jeden Desk wird dann ein eigenes Grafikfenster geöffnet. Die verschiedenen Desks können unabhängig voneinander angesprochen werden.

4.1.2 Funktionen der Klasse “desk”

```
open(int d,char *Name="winmagra",char *PosSize=NULL);
```

Diese Funktion öffnet die einzelnen Ausgabegeräte:

`int d:  SCREEN, EPS_FILE, PLT_FILE, ALL`

Die Ausgabegeräte können durch ODER (|) miteinander verknüpft werden. Außerdem können die Ausgabegeräte durch ODER (|) mit

`DOUBLE_SIZE`

verknüpft werden, was die jeweiligen Geräte in doppelter Höhe öffnet. Das Ausgabegerät ‘`SCREEN`’ kann durch ODER (|) mit

`MAXIMIZE`

verknüpft werden, wodurch der Desk als Vollbild auf dem Bildschirm geöffnet wird.

Der Parameter '`char *Name`' ist optional und legt den Namen der Ausgabe fest. Der default–Wert ist 'winmagra'. Der Parameter '`char *PosSize`' legt Position und Größe des Desks auf dem Bildschirm fest und hat die Form `"pos=(x,y) size=(width,height)"`. Die Koordinaten (0,0) bezeichnen dabei die linke obere Ecke des Bildschirms. Der Parameter selbst wie auch seine einzelnen Teile sind optional und werden beim Weglassen mit default–Werten belegt.

Beispiel: `dvi.open(SCREEN|EPS_FILE,"test1");`

Die Arbeitsfläche vom Typ 'desk' mit Namen 'dvi' (muss bereits existieren!) wird auf dem Bildschirm und als Postscript–File geöffnet, die Ausgabe trägt den Namen 'test1', bzw. 'test1.eps'.

Beispiel: `dvi.open(SCREEN|EPS_FILE,"test1","pos=(100,200) size=(300,400)");`

Die Arbeitsfläche vom Typ 'desk' mit Namen 'dvi' (muss bereits existieren!) wird auf dem Bildschirm und als Postscript–File geöffnet, die Ausgabe trägt den Namen 'test1', bzw. 'test1.eps'. Die linke obere Ecke des Desks wird gegenüber der linken oberen Ecke des Bildschirms um 100 Pixel nach rechts und 200 Pixel nach unten verschoben. Der Desk hat eine Breite von 300 Pixeln und eine Höhe von 400 Pixeln.

Beispiel: `dvi.open(SCREEN|MAXIMIZE);`

Die Arbeitsfläche vom Typ 'desk' mit Namen 'dvi' (muss bereits existieren!) wird als Vollbild auf dem Bildschirm geöffnet, die Ausgabe trägt den Namen 'winmagra'.

`void close();`

Schließt die Ausgabegeräte.

Aufruf: `dvi.close();`

`void clearviewport(ClearFlag f=CLEAR);`

Diese Funktion dient zum Löschen von Objekten auf der Arbeitsfläche, wobei das Verhalten durch den Parameter

```
ClearFlag f:  CLEAR, RESET, TERMINATE
```

gesteuert wird. Dabei löscht '`CLEAR`' Inhalt und Historie aller unabhängig von Boxen angelegter Objekte von der Arbeitsfläche, während '`RESET`' lediglich die Historie löscht. Im letzteren Fall wird also der momentane Inhalt der Arbeitsfläche bis zum nächsten Neuaufbau des Grafikfensters beibehalten. Die Boxen selbst samt Inhalt und Labels bleiben in beiden Fällen erhalten.

Der Aufruf

```
dvi.clearviewport();
```

entspricht dem Aufruf mit dem Parameter '`TERMINATE`' und löscht alles von der Arbeitsfläche, also auch die Boxen selbst.

`void setdeskposition(int x,int y);`

Setzt die Bildschirmkoordinaten der linken oberen Ecke des Desks. Diese sollten so gewählt werden, dass der Desk auch wirklich auf dem Bildschirm bleibt! Die Koordinaten (0,0) bezeichnen dabei die linke obere Ecke des Bildschirms. Der Aufruf dieser Funktion kann irgendwann nach dem Öffnen des Desks erfolgen.

Beispiel: `dvi.setdeskposition(200,400);`

Die linke obere Ecke des Desks wird gegenüber der linken oberen Ecke des Bildschirms um 200 Pixel nach rechts und 400 Pixel nach unten verschoben.

`void setcolor(int co);`

Setzt die Farbe des Zeichenstiftes.

Beispiel: `dvi.setcolor(_RED);`

`void setpenwidth(float p);`

Setzt die Breite des Zeichenstiftes mit ‘`float p`’ $\in [1, 6]$.

Beispiel: `dvi.setpenwidth(2.5);`

`float getpenwidth();`

Ermittelt die Breite des Zeichenstiftes.

Aufruf z.B.: `float b=dvi.getpenwidth();`

Wird bei den folgenden Funktionen die Farbe ‘`int co`’ nicht angegeben, wird die momentan gewählte Standardfarbe oder ‘`_BLACK`’ verwendet.

Die Ausdehnung der Arbeitsfläche geht normalerweise von $x = 0, y = 0$ (links unten) bis $x = 133, y = 100$ (rechts oben). Bei Angabe des Parameters ‘`DOUBLE_SIZE`’ liegt rechts oben bei $x = 133, y = 200$.

`void pset(float x,float y,int co=-1);`

Zeichnet mit der Farbe ‘`co`’ einen Punkt an die Stelle (x,y).

`void move(float x,float y);`

Bewegt den Zeichenstift zur Stelle (x,y).

`void draw(float x1,float y1,int co=-1);`

Zeichnet mit der Farbe ‘`co`’ eine Linie von der momentanen Stelle bis zum Punkt (x1,y1).

`void line(float x1,float y1,float x2,float y2,int co=-1);`

Zeichnet mit der Farbe ‘`co`’ eine Linie vom Punkt (x1,y1) bis zum Punkt (x2,y2).

`void rec(float x1,float y1,float x2,float y2,int co=-1);`

Zeichnet mit der Farbe ‘`co`’ ein Rechteck mit der Diagonalen (x1,y1), (x2,y2).

`void vector(float x1,float y1,float x2,float y2,int co=0,float sc=1);`

Zeichnet mit der Farbe ‘`co`’ einen Vektor mit Spitzengröße ‘`sc`’ vom Punkt (x1,y1) bis zum Punkt (x2,y2).

`void showbmp(float x1,float y1,float x2,float y2,char file[]);`

Lädt ein Bitmap aus einem File und stellt es im Rechteck (x1,y1,x2,y2) dar. Die Erweiterung ‘.bmp’ wird automatisch ergänzt.

Beispiel: `dvi.showbmp(30,30,90,90,"bild");`

`void putbmp(float x1,float y1,float x2,float y2,bitmap &bmp);`

> Zeichnet das Bitmap 'bmp' (ein Object der WinMagra–Klasse 'bitmap', s. später) in das Rechteck (x1,y1,x2,y2).
>
> Beispiel: `dvi.putbmp(30,30,90,90,bmp);`

`void showeps(char *fileName);`

> Importiert ein EPS–File und stellt es auf der Arbeitsfläche dar. Die Erweiterung '.eps' wird automatisch ergänzt.

Die folgende Funktion ist nur nötig, wenn sich das File **MAGFONT.BIN** mit den WinMagra–Fonts nicht im Standardpfad "C:\grafik\winmagra\lib\" befindet, und wird deshalb vom Benutzer normalerweise nicht verwendet.

`void setfontpath(char path[]);`

> Setzt den Pfad für die WinMagra–Fonts.
>
> Beispiel: `dvi.setfontpath("C:\winmagra\fonts");`

4.2 Klasse "boxclass"

Die "boxclass"–Klasse ermöglicht es, auf der Arbeitsfläche mit Diagrammen zu arbeiten. Sie sorgt für die korrekte Darstellung der Diagramme und der Daten. Eine Box kann nur in Zusammenhang mit einer Arbeitsfläche erzeugt und benutzt werden!

4.2.1 Konstruktor

`boxclass(desk &dvi);`

> Hierdurch wird eine Box vom Typ 'boxclass' auf einer Arbeitsfläche vom Typ 'desk' erzeugt. Es muss also bereits eine Arbeitsfläche existieren.
>
> Beispiel: `boxclass box(dvi);`
>
> Eine Box mit Namen 'box' wird auf einer Arbeitsfläche vom Typ 'desk' mit Namen 'dvi' erzeugt.

Auf einer Arbeitsfläche können mehrere Boxen vom Typ 'boxclass' erstellt werden. Die verschiedenen Boxen können unabhängig voneinander angesprochen werden.

4.2.2 Funktionen der Klasse "boxclass"

`void init(float x1,float y1,float x2,float y2,float zx=1,float zy=1);`

> Diese Funktion initialisiert die Box und positioniert sie als Rechteck mit der Diagonalen (x1,y1), (x2,y2) auf der Arbeitsfläche.
> Mit '`zx`' und '`zy`' kann der Abstand der minimalen und maximalen benutzerdefinierten Koordinaten (s.u.) vom Beginn und Ende der jeweiligen Achse eingestellt werden. Dabei sind normalerweise nur Werte zwischen 0.9 und 1 sinnvoll. Für 1 (Standardwert; muss nicht angegeben werden) fallen die Koordinatengrenzen mit den Achsgrenzen zusammen, für Werte größer 1 liegen die Extremalkoordinaten außerhalb der Achsen, für Werte kleiner 1 entsprechend innerhalb.
>
> Beispiel: `box.init(30,20,110,80,.96,.96);`
>
> Die Box mit der Diagonalen (30,20),(110,80) wird auf der Arbeitsfläche geöffnet, die Extremalkoordinaten sind um 4% gegenüber den Achsgrenzen eingerückt.

`void setaccuracy(float acc_x,float acc_y)`

Hiermit kann die Genauigkeit der Achsbeschriftung eingestellt werden: ‘0.1’ bedeutet eine Nachkommastelle, ‘0.01’ zwei, usw. Bei Nichtverwendung wird die Genauigkeit automatisch berechnet.

`void setAxisModulo(Achse a,int mod)`

Setzt für die angegebene Achse

```
Achse a:  boxclass::XACHSE, boxclass::YACHSE
```

einen Beschriftungsmodulo, in diesem Fall hat die Achsbeschriftung immer den Typ ‘Integer’.

Beispiel: `box.setAxisModulo(boxclass::XACHSE,24);`

setzt für die x–Achse eine 24–Stunden Beschriftung.

```
void dim(float xmin,float xmax,float ymin,float ymax,float xtic,float ytic,
     float tickscale=1,TICKSIDE t=INTICKS);
```

Hiermit werden die Achsen der Box dimensioniert, also das benutzerdefinierte Koordinatensystem eingestellt, und zwar von ‘`xmin`’ bis ‘`xmax`’, bzw. von ‘`ymin`’ bis ‘`ymax`’. Außerdem können an den Achsen Ticks im Abstand ‘`xtic`’ von ‘`xmin`’ und ‘`ytic`’ von ‘`ymin`’ gesetzt werden. Für xtic = ytic = 0 werden die Abstände automatisch berechnet, für xtic = ytic = −1 werden die Ticks unterdrückt. Mit ‘`tickscale`’ kann die Länge der Ticks geändert werden, und

```
TICKSIDE t:  boxclass::INTICKS, boxclass::OUTTICKS, boxclass::INOUTTICKS
```

setzt die Ticks in der Box, außerhalb der Box, oder an beiden Seiten.

Beispiel: `box.dim(1,10,-4,4,0,0);`

definiert ein Koordinatensystem von 1 bis 10 entlang der x-Achse, und von −4 bis 4 entlang der y-Achse. Die Ticks werden automatisch gesetzt, und zwar mit der Länge 1 nach innen.

Beispiel: `box.dim(1,10,-4,4,2,1,2,boxclass::INOUTTICKS);`

Diesmal werden Ticks im Abstand 2 entlang der x–Achse und im Abstand 1 entlang der y–Achse gesetzt. Die Ticks haben die Länge 2 und gehen nach innen und außen.

```
void label(char lx[]="",char ly[]="",float s=0.8,int x=0,
       FONT font=ROMAN,FONT tickFont=ARIAL);
```

Mit dieser Funktion werden die Achsen der Box beschriftet. ‘`s`’ gibt die Schriftgröße an, mit ‘`x`’ kann der Abstand der Beschriftung der y–Achse von der x–Achse geändert werden. Mit ‘`font`’ kann der Font für die Achsbeschriftung und mit ‘`tickFont`’ der Font für die Zahlen an den Achsen eingestellt werden.

Wird die Funktion ohne Parameter aufgerufen, oder hat eine der Achsbeschriftungen die Länge 0, so werden auch die Zahlen an der jeweiligen Achse unterdrückt. Um zwar Zahlen, aber keine Schrift an der Achse zu erhalten, muss als String z.B. `" "`, also mindestens ein Leerzeichen übergeben werden.

Beispiel: `box.label("x-Achse","y-Achse");`

Die Achsen erhalten die Beschriftungen ‘x-Achse’ und ‘y-Achse’ im Standardformat.

Beispiel: `box.label("","y-Achse",1,2);`

Die x-Achse erhält keine Beschriftung und auch keine Zahlen. Die Schriftgröße wurde auf ‘1’ festgesetzt, und ist damit etwas größer als der Standartwert. Die Beschriftung der y–Achse rutscht etwas weiter von der x–Achse weg.

```
void clear(ClearFlag f=CLEAR);
```

Diese Funktion dient zum Löschen des Inhaltes der Box, wobei das Verhalten durch den Parameter

```
ClearFlag f:  CLEAR, RESET
```

gesteuert wird. Dabei löscht 'CLEAR' den Inhalt und die Historie der Box, während 'RESET' lediglich die Historie löscht. Im letzteren Fall wird also der momentane Inhalt der Box bis zum nächsten Neuaufbau des Grafikfensters beibehalten.

Der Aufruf

```
box.clear();
```

entspricht dem Aufruf mit dem Parameter 'CLEAR'.

Ein 'RESET' oder 'CLEAR' ist immer dann nötig, wenn viele Zeichnungen in Form einer Animation hintereinander gezeigt werden sollen, da es sonst zu einem Speicherüberlauf kommen kann.

Alle mit den folgenden Funktionen erzeugten Linien, Punkte, Schriften, Symbole etc. werden grundsätzlich nur *innerhalb* der Box gezeichnet!

```
void showpoints(int x[],int y[],int n,enum COLOR co=_BLACK,
                enum SYMBOL sy=SQUARE,float s=1);
void showpoints(float x[],float y[],int n,enum COLOR co=_BLACK,
                enum SYMBOL sy=SQUARE,float s=1);
void showpoints(long double x[],long double y[],int n,enum COLOR co=_BLACK,
                enum SYMBOL sy=SQUARE,float s=1);
```

Der durch 'x[]' und 'y[]' repräsentierte Datensatz der Länge 'n' wird durch Symbole 'sy' der Größe 's' mit der Farbe 'co' dargestellt.

Minimalaufruf: `box.showpoints(a,b,n);`

Der Datensatz 'a[]', 'b[]' der Länge 'n' wird durch schwarze Quadrate der Größe '1' dargestellt.

```
void showerrorbars(int x[],int y[],int err[],int n,
                   enum SYMBOL sy,enum COLOR co=_BLACK,float s=1);
void showerrorbars(float x[],float y[],float err[],int n,
                   enum SYMBOL sy,enum COLOR co=_BLACK,float s=1);
void showerrorbars(long double x[],long double y[],long double err[],int n,
                   enum SYMBOL sy,enum COLOR co=_BLACK,float s=1);
```

Der durch 'err[]' repräsentierte Fehler–Datensatz der Länge 'n' wird als Fehlerbalken am Datensatz 'x[], y[]' mit der Farbe 'co' dargestellt. Als Symbole sind hier nur 'XBARS' für Fehlerbalken in x–Richtung bzw. 'YBARS' für Fehlerbalken in y–Richtung sinnvoll.

```
void connectpoints(int x[],int y[],int n,enum COLOR co=_BLACK,float sk=1);
void connectpoints(float x[],float y[],int n,enum COLOR co=_BLACK,float sk=1);
void connectpoints(long double x[],long double y[],int n,enum COLOR co=_BLACK,
                   float sk=1);
```

Der durch 'x[]' und 'y[]' repräsentierte Datensatz der Länge 'n' wird als Linie mit Unterbrechungsgrad 'sk' mit der Farbe 'co' dargestellt. 'sk = 1' ergibt eine durchgezogene und 'sk < 1' eine unterbrochene Linie.

```
void showhistogram(int x[],int y[],int n,enum COLOR co=_BLACK);
void showhistogram(float x[],float y[],int n,enum COLOR co=_BLACK);
void showhistogram(long double x[],long double y[],int n,enum COLOR co=_BLACK);
```

Der durch '`x[]`' und '`y[]`' repräsentierte Datensatz der Länge '`n`' wird als Histogramm mit durchgezogenen Linien mit der Farbe '`co`' dargestellt.

```
void vector(float xa,float ya,float xe,float ye,enum COLOR co=0,float sc=1);
```

Zeichnet mit der Farbe '`co`' einen Vector mit Spitzengröße '`sc`' vom Punkt (xa,ya) bis zum Punkt (xe,ye).

```
void showvectors(float x[],float y[],int n,enum COLOR co=0,float sc=1);
```

Der durch '`x[]`' und '`y[]`' repräsentierte Datensatz der Länge '`n`' wird als Vektorlinie mit der Spitzengröße '`sc`' mit der Farbe '`co`' dargestellt.

```
void pset(float x,float y,int co=0);
```

Zeichnet mit der Farbe '`co`' einen Punkt an die Stelle (x,y).

```
void setsymbol(float x,float y,enum COLOR co=_BLACK,enum SYMBOL sy=SQUARE,float s=1);
```

Zeichnet das Symbol '`sy`' mit der Farbe '`co`' und der Größe '`s`' an die Stelle (x,y).

```
void line(float xa,float ya,float xe,float ye,enum COLOR co=_BLACK,float sk=1);
```

Zeichnet mit der Farbe '`co`' eine Linie mit Unterbrechungsgrad '`sk`' vom Punkt (xa,ya) bis zum Punkt (xe,ye). '$sk = 1$' ergibt eine durchgezogene und '$sk < 1$' eine unterbrochene Linie.

```
void rec(float xa,float ya,float xe,float ye,enum COLOR co=_BLACK);
```

Zeichnet mit der Farbe '`co`' ein Rechteck mit der Diagonalen (x1,y1), (x2,y2).

```
void showbmp(float xa,float ya,float xe,float ye,char file[]);
```

Lädt ein Bitmap aus einem File und stellt es im Rechteck (xa,ya,xe,ye) dar. Die Erweiterung '.bmp' wird automatisch ergänzt.

Beispiel: `box.showbmp(0,0,256,256,"bild");`

```
void hwrite(float x,float y,char s[],enum COLOR co=_BLACK,int fo=1,float sc=1,
        float tjx=1,float tjy=1);
void vwrite(float x,float y,char s[],enum COLOR co=_BLACK,int fo=1,float sc=1,
        float tjx=1,float tjy=1);
```

Schreibt den Text '`s`' horizontal (hwrite), bzw. vertikal nach oben (vwrite) mit der Farbe '`co`', der Schriftart '`fo`' und der Größe '`sc`' an die Stelle (x,y). Mit Hilfe von '`tjx`' und '`tjy`' kann der Text relativ zu (x,y) plaziert werden:

(tjx = 1,tjy = 1): (x,y) ist die Koordinate der linken unteren Ecke des Textes
(tjx = 0,tjy = 0): (x,y) ist die Koordinate der Mitte des Textes
(tjx = −1,tjy = −1): (x,y) ist die Koordinate der rechten oberen Ecke des Textes

Minimalaufruf: `box.hwrite(5,10,"test");`

Der Text '`test`' wird bei der 'box'-Koordinate (5,10) beginnend horizontal im Font '`ROMAN`' mit schwarzer Farbe und Größe 1 geschrieben.

4.3 Klasse “fontclass”

Die “fontclass”-Klasse arbeitet normalerweise im Hintergrund und ist für das Verwalten und Darstellen der verschiedenen Schriftarten zuständig. In den meisten Anwendungen wird der Benutzer die Klasse nicht direkt verwenden, es sei denn, es soll Schrift unabhängig von einer Box dargestellt werden, wenn also z.B. eine zusätzliche Beschriftung außerhalb der Box benötigt wird.

Ein eigenständiger Font kann nur in Zusammenhang mit einer Arbeitsfläche vom Typ “desk” erzeugt und benutzt werden! Eine Übersicht aller druckbarer Zeichen ist im Anhang angegeben.

4.3.1 Konstruktor

```
fontclass(desk &dvi);
```

Hierdurch wird ein Font vom Typ ‘fontclass’ auf einer Arbeitsfläche vom Typ ‘desk’ erzeugt. Es muss also bereits eine Arbeitsfläche existieren.

Beispiel: `fontclass font(dvi);`

Ein Font mit Namen ‘font’ wird auf einer Arbeitsfläche vom Typ ‘desk’ mit Namen ‘dvi’ erzeugt.

4.3.2 Funktionen der Klasse “fontclass”

```
void hwrite(float x,float y,char s[],int co=0,int fo=1,float sc=1,
            float tjx=1,float tjy=1);
void vwrite(float x,float y,char s[],int co=0,int fo=1,float sc=1,
            float tjx=1,float tjy=1);
```

Schreibt den Text ‘`s`’ horizontal (hwrite), bzw. vertikal nach oben (vwrite) mit der Farbe ‘`co`’, der Schriftart ‘`fo`’ und der Größe ‘`sc`’ an die Stelle (x,y). Mit Hilfe von ‘`tjx`’ und ‘`tjy`’ kann der Text relativ zu (x,y) plaziert werden:

(tjx $= 1$,tjy $= 1$): (x,y) ist die Koordinate der linken unteren Ecke des Textes
(tjx $= 0$,tjy $= 0$): (x,y) ist die Koordinate der Mitte des Textes
(tjx $= -1$,tjy $= -1$): (x,y) ist die Koordinate der rechten oberen Ecke des Textes

Minimalaufruf: `font.hwrite(5,10,"test");`

Der Text ‘`test`’ wird bei der ‘desk’-Koordinate (5,10) beginnend horizontal im Font ‘`ROMAN`’ mit schwarzer Farbe und Größe 1 geschrieben.

4.4 Klasse “symbol”

Die “symbol”-Klasse arbeitet normalerweise im Hintergrund und ist für das Verwalten und Darstellen der verschiedenen Symbole zuständig. In den meisten Anwendungen wird der Benutzer die Klasse nicht direkt verwenden, es sei denn, es soll ein Symbol unabhängig von einer Box dargestellt werden, wenn also z.B. ein zusätzliches Symbol außerhalb der Box benötigt wird.

Ein eigenständiges Symbol kann nur in Zusammenhang mit einer Arbeitsfläche vom Typ “desk” erzeugt und benutzt werden!

4.4.1 Konstruktor

```
symbol(desk &dvi);
```

Hierdurch wird ein Objekt vom Typ ‘symbol’ auf einer Arbeitsfläche vom Typ ‘desk’ erzeugt. Es muss also bereits eine Arbeitsfläche existieren.

Beispiel: `symbol sym(dvi);`

Ein ‘symbol’–Objekt mit Namen ‘sym’ wird auf einer Arbeitsfläche vom Typ ‘desk’ mit Namen ‘dvi’ erzeugt.

4.4.2 Funktionen der Klasse “symbol”

```
void put(float x,float y,enum SYMBOL sy=NONE,enum COLOR co=_BLACK,float s=1);
```

Zeichnet das Symbol ‘`sy`’ mit der Farbe ‘`co`’ und der Größe ‘`s`’ an die Stelle (x,y).

Minimalaufruf: `sym.put(5,10,TRIANGLE);`

Bei den ‘desk’-Koordinaten (5,10) wird ein schwarzes Dreieck der Größe 1 gezeichnet.

4.5 Klassen “bitmap” und “bitmap24”

Mit Hilfe dieser Klassen können Bitmaps eingelesen, gespeichert, dargestellt und bearbeitet werden. Die Klassen sind für die korrekte Verwaltung der Daten verantwortlich. Die “bitmap”–Klasse ist für schwarzweiß Bitmaps mit 256 Graustufen zuständig, die “bitmap24”–Klasse ist das entsprechende Gegenstück für Farb–Bitmaps.

Die “bitmap”–Klassen sind zunächst von einer grafischen Darstellung unabhängig und können auch ohne ‘desk’ oder ‘boxclass’ nur zum Manipulieren von Bitmaps verwendet werden.

4.5.1 Konstruktor

```
bitmap(int b,int h);
bitmap24(int b,int h);
```

Hierdurch wird ein Objekt vom Typ ‘`bitmap`’ bzw. ‘`bitmap24`’ mit der Breite ‘`b`’ und der Höhe ‘`h`’ erzeugt.

Beispiel: `bitmap bmp(1024,1024);`

Ein ‘bitmap’–Objekt mit Namen ‘bmp’ wird mit einer Höhe und Breite von je 1024 erzeugt.

Sollen so erzeugte Bitmaps auf einem ‘desk’ dargestellt werden (mit oder ohne Box), so ist es dringend empfehlenswert, zuerst das ‘bitmap’–Objekt und dann das ‘desk’–Objekt zu erzeugen.

4.5.2 Funktionen der “bitmap”–Klassen

```
void load(char name[]);
```

Liest das Bitmap aus einem File mit Namen ‘`name[]`’, die Erweiterung ‘.bmp’ wird automatisch ergänzt.

Beispiel: `bmp.load("test");`

Das im File mit Namen ‘test.bmp’ enthaltene Bitmap wird nach ‘bmp’ geladen.

`void save(char name[]);`

Speichert das Bitmap in ein File mit Namen '`name[]`', die Erweiterung '.bmp' wird automatisch ergänzt.

Beispiel: `bmp.save("test");`

Das in 'bmp' enthaltene Bitmap wird in ein File mit Namen 'test.bmp' geschrieben.

```
int getwidth();
int getheight();
```

Ermittelt die Breite, bzw. die Höhe des Bitmaps.

Beispiel: `int breite=bmp.getwidth();`

4.5.3 Operatoren der "bitmap"–Klasse

`unsigned char* operator[](int y);`

Dieser Operator ermöglicht den Zugriff auf die einzelnen Bildpunkte, so greift '`bmp[y][x]`' auf den Punkt (x,y) zu. Die Punkte können gelesen und geschrieben werden.

Beispiel: `bmp[12][124]=13;`

Dem 124. Punkt der 12. Zeile wird der Wert 13 zugeordnet.

Beispiel: `int i=bmp[12][124];`

Der Wert des 124. Punktes der 12. Zeile wird in die Variable '`i`' eingelesen.

4.5.4 Operatoren der "bitmap24"–Klasse

`unsigned char* operator()(short y, short x);`

Dieser Operator ermöglicht den Zugriff auf die einzelnen Bildpunkte, so greift '`bmp(y,x)[enum CHANEL ch]`' auf den Farbkanal '`ch`' des Punktes (x,y) zu. Der Farbkanal kann aus den Farben

```
enum CHANEL ch:  BLUE, GREEN, RED
```

ausgewählt werden. Die Punkte können gelesen und geschrieben werden.

Beispiel: `bmp24(12,124)[bitmap24::RED]=13;`

Der roten Farbe des 124. Punktes der 12. Zeile wird der Wert 13 zugeordnet.

Beispiel: `int i=int(bmp24(12,124)[bitmap24::RED]);`

Der Wert der roten Farbe des 124. Punktes der 12. Zeile wird in die Variable '`i`' eingelesen.

4.6 Die "GetData"–Klassen

Die Klasse "GetData" ist als Template angelegt und dient zum Lesen von Daten aus Dateien. Der Vorteil der Template–Deklaration liegt darin, dass die Klasse als Objekt für jeden Datentyp angelegt werden kann. Derzeit ist "GetData" für integer–Daten ("GetIntData"–Klasse), für float–Daten ("GetFloatData"–Klasse) und für double–Daten ("GetDoubleData"–Klasse) implementiert.

Ein Objekt der jeweiligen Klasse wird, analog zu anderen Klassen, durch einen Konstruktor erzeugt. Die mit Daten geladene Klasse enthält für die X– und Y–Achse jeweils ein Array, welches die jeweiligen Daten beinhaltet. Auf die Daten kann mittels eines zurückgegebenen Pointers zugegriffen werden. Die Kommunikation mit der Klasse erfolgt über Operatoren und Funktionen.

Die "GetData"–Klassen sind zunächst von einer grafischen Darstellung unabhängig und können auch ohne 'desk' oder 'boxclass' nur zum Einlesen von Daten verwendet werden.

4.6.1 Konstruktor

```
GetIntData();
GetFloatData();
GetDoubleData();
```

Hierdurch wird ein Objekt vom jeweiligen Typ erzeugt.

Beispiel: `GetFloatData GFD;`

Ein Objekt mit Namen ‘`GFD`’ vom Typ ‘`GetFloatData`’ wird erzeugt.

4.6.2 Beispiel zur Benutzung

```
...
int *X,*Y;
GetIntData GID;
    GID("test.dat","u 1:3");
    X=GID(GetIntData::XACHSE);
    Y=GID(GetIntData::YACHSE);
...
```

Die Daten aus dem File mit Namen ‘test.dat’ werden durch das “GetIntData”–Objekt mit Namen ‘GID’ eingelesen. Dabei werden die Daten der 1. Spalte intern als ‘`XACHSE`’ und die der 3. Spalte als ‘`YACHSE`’ verwaltet. Danach werden diese Daten den integer–Pointern ‘`X`’ und ‘`Y`’ zugeordnet.

4.6.3 Operatoren der “GetData”–Klassen

`int operator()(char *fileName);`

Dieser Operator erwartet als Argument den Namen der Datei aus der die Daten gelesen werden sollen. Die Daten der 1. Spalte werden als ‘`XACHSE`’ und die der 2. Spalte als ‘`YACHSE`’ gelesen. Zeilen, die mit ‘`#`’ beginnen, werden dabei als Kommentarzeilen interpretiert und deshalb nicht eingelesen.

`int operator()(char *fileName,char *use);`

Dieser Operator erwartet als erstes Argument den Namen der Datei aus der die Daten gelesen werden sollen und als zweites Argument die Zuweisung der Spalten oder Zeilen zu den Achsen. Dabei wird mit `"u a:b"` die Spalte Nummer ‘`a`’ der ‘`XACHSE`’ und die Spalte Nummer ‘`b`’ der ‘`YACHSE`’ zugeordnet. Mit `"z a:b"` dagegen wird die Zeile Nummer ‘`a`’ der ‘`XACHSE`’ und die Zeile Nummer ‘`b`’ der ‘`YACHSE`’ zugeordnet. Zeilen, die mit ‘`#`’ beginnen, werden dabei als Kommentarzeilen interpretiert und deshalb nicht eingelesen.

Beispiel: `GFD("test.dat","z 4:1");`

Die Daten aus dem File mit Namen ‘test.dat’ werden durch das Objekt ‘GFD’ eingelesen. Dabei werden die Daten der 4. Zeile intern als ‘`XACHSE`’ und die der 1. Zeile als ‘`YACHSE`’ verwaltet.

`<Datentyp> *operator()(FLAG f);`

Dieser Operator gibt einen Pointer vom Typ der jeweiligen Klasse für das mit ‘`f`’ angeforderte Array zurück. ‘`f`’ kann dabei folgende Werte annehmen:

```
FLAG f:  XACHSE, YACHSE
```

Beispiel: `float *X=GFD(GetFloatData::XACHSE);`

Der float–Pointer ‘`*X`’ zeigt auf die durch das Objekt ‘GFD’ als ‘`XACHSE`’ eingelesenen Daten.

4.6.4 Funktionen der “GetData”–Klassen

```
unsigned short getZeilen();
unsigned short getSpalten();
```

Diese Funktionen ermitteln die Anzahl der lesbaren, unkommentierten Zeilen, bzw. Spalten des eingelesenen Datenfiles.

```
unsigned short getDataLength();
```

Diese Funktion ermittelt die Anzahl der eingelesenen Daten pro Achse.

Beispiel: `int len=GFD.getDataLength();`

```
<Datentyp> getXMIN();
<Datentyp> getXMAX();
```

Diese Funktionen ermitteln Minimum und Maximum der für die ‘`XACHSE`’ eingelesenen Daten.

Beispiel: `float xmi=GFD.getXMIN();` `float xma=GFD.getXMAX();`

```
<Datentyp> getYMIN();
<Datentyp> getYMAX();
```

Diese Funktionen ermitteln entsprechend Minimum und Maximum der für die ‘`YACHSE`’ eingelesenen Daten.

Kapitel 5

Weitere Funktionalitäten

5.1 Anzeigen der Koordinaten

Durch Klicken der linken Maustaste können die Koordinaten der momentanen Position des Mauszeigers ermittelt werden.

Ein Linksklick innerhalb der Box öffnet im Grafikfenster links oben ein kleines Fenster. Dieses zeigt die Koordinaten in Boxeinheiten an, solange sich der Mauszeiger in der Box befindet. Ein zweiter Linksklick schließt das Fenster wieder.

Ein Linksklick auf dem Desk außerhalb der Box öffnet unterhalb des Boxkoordinaten–Fensters ein weiteres kleines Fenster. Dieses zeigt die Koordinaten in Deskeinheiten an, und zwar für alle Positionen des Mauszeigers, also innerhalb und außerhalb der Box. Ein zweiter Linksklick schließt das Fenster wieder.

Es können auch beide Fenster gleichzeitig geöffnet sein.

5.2 Lupe

Zum Vergrößern der grafischen Darstellung stehen Lupen mit 2-facher, 4-facher und 8-facher Vergrößerung zur Verfügung.

Die 2-fache Vergrößerung wird durch Drücken der rechten Maustaste aktiviert. Die 4-fache Vergrößerung wird durch Drücken der mittleren Maustaste aktiviert, bzw. durch Drücken der rechten Maustaste bei gedrückter Shift–Taste (wenn z.B. keine mittlere Maustaste zur Verfügung steht). Die 8-fache Vergrößerung wird durch Drücken der mittleren Maustaste bei gedrückter Shift–Taste aktiviert, bzw. durch Drücken der rechten Maustaste bei gedrückter Ctrl– (= Strg–) Taste.

Mit der Maus kann nun die Grafik in der jeweiligen Vergrößerung abgefahren werden, wobei die zuständige Maustaste immer gedrückt zu halten ist. Die Shift– oder Ctrl–Taste kann dabei losgelassen werden.

5.3 Drucken

Neben dem Umweg über ein Postscript–File (Angabe des Parameters '`EPS_FILE`' beim Öffnen des Ausgabegerätes, s. Abschnitt 4.1.2) können WinMagra–Grafiken auch direkt aus dem Grafikfenster heraus gedruckt werden.

Durch Klicken der linken Maustaste auf dem WinMagra–Symbol am linken Ende der Titelleiste, bzw. durch Klicken der rechten Maustaste irgendwo in der Titelleiste wird eine Menü geöffnet. Mit Hilfe des Menüpunktes 'Drucker einrichten ...' kann ein Drucker ausgewählt und konfiguriert werden und mit Hilfe des Menüpunktes 'Drucken' kann die Grafik dann auf diesem Drucker ausgedruckt werden.

Anhang A

Die WinMagra Fonts

A.1 Tabelle aller druckbarer Zeichen

Die folgende Grafik zeigt alle mit der WinMagra druckbaren Zeichen:

Font ARIAL:	abcdefghijklmnopqrstuvwxyz	
Font ROMAN:	**abcdefghijklmnopqrstuvwxyz**	
Font GREEK:	$\alpha\beta\eta\delta\varepsilon\varphi\gamma\chi\iota\theta\kappa\lambda\mu\nu o\pi\ \rho\sigma\tau\vartheta\psi\omega\xi\upsilon\zeta$	
Font ARIAL:	ABCDEFGHIJKLMNOPQRSTUVWXYZ	
Font ROMAN:	**ABCDEFGHIJKLMNOPQRSTUVWXYZ**	
Font GREEK:	ΑΒΗΔΕΦΓΧΙΘΚΛΜΝΟΠ ΡΣΤ ΨΩΞΥΖ	
Font ARIAL:	1234567890	! " $ % & / ( ) =
Font ROMAN:	**1234567890**	**! " $ % & / ( ) =**
Font GREEK:	1234567890	$\surd$ | ∇ $\pm$ $\propto$ ℃ $\div$ $\sim$ =
Font ARIAL:	ä ö ü Ä Ö Ü ß ?	ä ⌢ _ ü ö ß ⌢
Font ROMAN:	**ä ö ü Ä Ö Ü ß ?**	**ä [ ] ü ö ß [**
Font GREEK:	$\leq$ ‾ $\geq$ $\parallel$ $\approx$ $\perp$ © $\neq$	$\leq$ $\leftarrow$ $\downarrow$ $\geq$ ‾ © $\leftarrow$
Font ARIAL:	‘ + * # ’ − _ . : , ; < >	
Font ROMAN:	**‘ + * # ’ −] . : , ; < >**	
Font GREEK:	§ + · © $\int$ − $\downarrow$ $\vec{}$ 2 ⌢ ~ $\ll\gg$	

Alle Zeichen können durch eine ODER–Verknüpfung (|) des jeweiligen Fonts mit '`SMALL`' oder '`BOLD`' auch dünn oder dick dargestellt werden.

Einzelne Textabschnitte können durch '`_{abc}`' oder '`^{abc}`' tief– oder hochgestellt werden. Durch '`$abc$`' wird der jeweilige Text im Font '`GREEK`' dargestellt. In diesem Font befinden sich z.B. auch mathematische Zeichen und verschiedene Sonderzeichen.

Zur besseren Nachvollziehbarkeit wurden bei den Sonderzeichen Leerzeichen eingefügt. Einige Zeichen müssen durch Voranstellen eines '`\`' oder '`/`' gedruckt werden. Die Einzelheiten können dem zugrunde liegenden Programm entnommen werden. Dieses hat den Namen "fonts.cpp" und befindet sich im Verzeichnis "C:\grafik\winmagra\beispiel".