PureBasic

Karsten, Sulfur, JonRe, Robert

de.wikibooks.org

26. August 2014

Inhaltsverzeichnis

1 Einstieg **3**
1.1 Über PureBasic 3

2 Grundlagen **5**
2.1 Hello, world! 5
2.2 Variablen und Konstanten 7
2.3 Variablentypen 10
2.4 Bedingungen 13
2.5 Schleifen 16
2.6 Arrays, Listen und Maps 19

3 Fortgeschrittene Themen **25**
3.1 Strukturen 25
3.2 Prozeduren 26
3.3 Code-Auslagerung 31
3.4 Zeiger 33
3.5 GUI-Programmierung 34
3.6 Event-Handler 41
3.7 SizeWindow 45
3.8 Zeichnen 46

4 Objektorientierung **51**
4.1 Installation von PureObject 51
4.2 Einstieg in die Objektorientierte Programmierung 52
4.3 Vererbung 54
4.4 Polymorphismen 55

5 Aufgabenlösungen **59**
5.1 Grundlagen-Lösungen 59
5.2 Fortgeschrittene Themen-Lösungen 62
5.3 Objektorientierung-Lösungen 63

6 Beispielprogramme **65**
6.1 sFTPc 65
6.2 reversePolishNotation 72

7 Autoren **75**

Abbildungsverzeichnis **77**

8 Licenses **81**
8.1 GNU GENERAL PUBLIC LICENSE 81
8.2 GNU Free Documentation License 82
8.3 GNU Lesser General Public License 83

1 Einstieg

1.1 Über PureBasic

PureBasic (PB) ist ein proprietärer Basic-Dialekt, der von Frédéric Laboureur im Jahre 2000 aus dem Beta-Status entlassen wurde. Die Sprache ging aus Bibliotheken hervor, die Laboreur in den 1990er Jahren für BlitzBasic auf dem Amiga entwickelte. Heute erhält man neben dem Compiler noch eine IDE und viele andere Werkzeuge, die das Programmieren vereinfachen. Eine beschränkte Demo-Version ist ebenfalls verfügbar.

PB ist für Windows, Linux und Mac OSX verfügbar und wird aktiv weiterentwickelt. Durch die klassisch-einfache Basic-Syntax ist PB einfach zu erlernen und verfügt trotzdem über Features, die für die fortgeschrittene und profesionelle Programmierung notwendig sind, wie z.B. Zeiger. Des Weiteren bietet PB viele Sprachelemente, um auf die API des jeweiligen Systems zuzugreifen, wodurch z.B. die GUI-Entwicklung vereinfacht wird. Inline-Assembler wird ebenfalls unterstützt. Weitere Features sollen hier dargestellt werden:

- Nutzung von Medien (z.B. CDs, MP3, AVI)
- Bibliothek für Sprites
- Integrierte 3D Engine (OGRE)
- Unterstützung von DLLs
- Verarbeitung von XML Dokumenten
- Bibliotheken für Netzwerkprogrammierung

Seit 2006 ist außerdem objektorientiertes Programmieren als Plugin verfügbar.

2 Grundlagen

2.1 Hello, world!

Die Tradition des "Hello, world!"-Programmes wurde von Brian Kernighan eingeführt, als er eine Dokumentation über die Programmiersprache B schrieb. Durch das Buch *The C Programming Language* erlang das Beispiel Bekanntheit.

"Hello, world!" ist hervorragendes Beispiel für jede Programmiersprache, da man mit jenem die wesentlichen Aspekte dieser erfassen und darlegen kann. Von daher ist es nur angebracht es hier ebenfalls zu verwenden.

```
; Listing 1: Hello, world!

OpenConsole()
PrintN("Hello, world!")
Delay(2000)
```

```
Ausgabe:

Hello, world!
```

Nachdem man PureBasic installiert hat, öffnet man die IDE über das Startmenü (Windows) bzw. über den Befehl *purebasic* (Linux & Mac OSX). Eine IDE ist eine Entwicklungsumgebung, in der alle Werkzeuge, die man zum Programmieren benötigt, organisiert und schnell verfügbar sind. Man gibt nun den oberen Codeabschnitt ein und drückt F5. Für dieses und alle anderen Kapitel soll gelten, dass man die Programme am Besten von Hand eingibt, da man durch simples Copy&Paste keine Programmiersprache lernt, sondern indem man selber schreibt und ausprobiert!

Hat man die Anleitung beachtet, müsste nun ein Fenster offen sein, in welchem *Hello, world!* steht und das sich nach 2 Sekunden wieder schließt. Unter Linux und Mac wird die Ausgabe direkt auf der Shell sichtbar, über die man die IDE öffnete. Warum geschieht dies? Die erste Zeile stellt einen sogenannten Kommentar dar, d.h. eine Befehlszeile, die vom sogenannten *Compiler* ignoriert wird. Dies sieht man an dem Semikolon, dass am Anfang der Zeile steht. Kommentare können überall im Code stehen, auch hinter einer Codezeile. Sie werden genutzt, um den Code *sinnvoll* zu kommentieren. Code soll für sich selber sprechen und Kommentare stellen eine Ergänzung dar, um sich in diesem einfacher zurecht zu finden. Der Compiler ist das Programm, dass den für den Programmierer lesbaren Code in eine für den Computer ausführbare Datei übersetzt. Nach dem Kommentar kommt der erste Befehl: *OpenConsole()*. Dieser tut nichts anderes, als unter Windows eine Konsole zu öffnen, unter Linux und Mac wird er ignoriert, da Konsolenprogramme, wie dieses eines ist hier immer

über die Shell gestartet werden, trotzdem muss er auch auf diesen Betriebssystemen immer geschrieben werden, wenn es sich um ein Konsolenprogramm handelt.

Es werden an dieser Zeile wesentliche Eigenschaften von PureBasic klar: Der PureBasic-Compiler arbeitet zeilenorientiert, d.h. pro Zeile steht ein Befehl. *PrintN("Hello, world!")* tut nichts anderes, als die Zeile *Hello, world!* auf die Ausgabe zu schreiben und danach einen Zeilenumbruch einzufügen. *"Hello, world!"* ist hierbei das Argument bzw. ein Parameter von *PrintN()*, also etwas, das von *PrintN()* zur Ausführung benötigt wird, in diesem Fall wird ein String (Zeichenkette) benötigt. Was das ist, wird in einem anderen Kapitel erklärt. *Delay(2000)* hält die Ausführung für 2000 Millisekunden, also 2 Sekunden, an. *2000* ist wieder ein Argument, diesmal jedoch ein Zahlenwert. Es fällt auf, dass das Argument von *Delay()* nicht in Anführungsstrichen steht. Dies liegt daran, dass *Delay()* einen Zahlenwert erwartet und keinen String wie *PrintN()*, denn Strings stehen in PureBasic immer in Anführungsstrichen, ansonsten werden sie vom Compiler nicht als solche interpretiert.

Nachdem nun klar ist wie das Programm funktioniert, sollte man ein bisschen mit diesem herumspielen, um sich mit der Syntax, also dem Aufbau eines PureBasic-Programms, vertraut zu machen. Was passiert zum Beispiel, wenn man die Anführungstriche bei *PrintN("Hello, world!")* weglässt? Man sollte sich mit den Ausgaben des Debuggers vertraut machen; diese helfen weiter, wenn etwas falsch programmiert wurde oder andere Probleme auftreten.

Wenn man Fragen zu einem Befehl hat, kann man auf diesen klicken und dann F1 drücken. So wird direkt die Hilfeseite zu diesem Befehl geöffnet. Ansonsten gelangt man über F1 in die allgemeine Referenz, die auch im Internet eingesehen werden kann. Der Link ist im Inhaltsverzeichnis verfügbar.

2.1.1 Der Debugger

Der Debugger zeigt, wie schon gesagt, Fehler an. Er wird durch das Fenster unter dem Editor repräsentiert. Lässt man z.B die Anführungszeichen bei *PrintN("Hello, world!")* weg, zeigt er beim Kompilieren an: "Zeile 4: Syntax-Fehler." Aus dieser Information kann man erschließen, dass das Programm nicht kompiliert werden konnte, weil in Zeile 4 etwas falsch ist und ausgebessert werden muss.

Der Debugger kann auch in der Entwicklung eines Projekts nützlich sein, denn mit ihm kann man die Ausführung des Programmes dokumentieren.

```
; Listing 2: Hello, world! mit Debugger

OpenConsole()
Debug "Jetzt kommt die Ausgabe"
PrintN("Hello, world!")
Debug "Nun wartet das Programm 2 Sekunden"
Delay(2000)
```

```
Ausgabe:

Hello, world!
```

Man drückt abermals auf F5, nachdem man das Programm eingegeben hat. Es kommt wieder die Ausgabe aus dem ersten Programm, jedoch passiert noch etwas anderes: Es öffnet sich ein Fenster in dem die Strings hinter *Debug* angezeigt werden. Die Strings sind hierbei aber keine Argumente, sondern sogenannte Ausdrücke, also etwas, das ausgewertet werden kann. *PrintN("Hello, world!")* ist ein Ausdruck, denn er kann zu einer Ausgabe von *Hello, world!* auf der Konsole ausgewertet werden. Es gibt noch viele weitere solcher Debugger-Schlüsselwörter. Das Wort "Schlüsselwörter" wird in einem anderen Kapitel erläutert. Das Interessante an diesen Schlüsselwörtern ist, dass sie nur bei aktivierten Debugger kompiliert werden. Dieser ist standardmäßig aktiviert. Sie stellen also eine großartige Hilfestellung bei der Projekterstellung dar, ohne dass man sich in der finalen Version darum kümmern muss alle Befehle zu entfernen.

Der Debugger kann über den Reiter *Debugger* deaktiviert werden, in der Demo-Version lässt er sich jedoch nicht ausschalten.

2.1.2 Aufgaben

1. Niemals vergessen: Programmieren lernt man nur durch selbst schreiben und ausprobieren, also auch durch bewusstes Fehler einbauen und Verändern des Codes.
2. Was ist an folgendem Programm falsch?

```
OpenConsole() PrintN(Hello, world!)
Delay(2000
```

2.2 Variablen und Konstanten

Variablen sind einfach gesagt Datenspeicher. In ihnen können sowohl Zahlenwerte als auch Zeichenketten (Strings) zur späteren Benutzung gespeichert werden.

```
; Listing 3: Variablen

OpenConsole()

x = 1
PrintN(Str(x))

x = 2
PrintN(Str(x))

y = x+1
PrintN(Str(y))

y + 1
PrintN(Str(y))

Delay(2000)
```

```
Ausgabe:

1
2
```

```
3
4
```

Nachdem das Programm wie üblich kompiliert wurde, sieht man die Zahlen 1, 2, 3 und 4 untereinander ausgegeben. Neu sind die Variablen. Variablen funktionieren an sich ganz einfach: Man *deklariert* und *definiert* sie, d.h. man teilt dem Compiler mit, dass es nun eine neue Variable mit einem Bezeichner gibt und ihr wird automatisch vom Compiler Speicherplatz zugewiesen. Das funktioniert an jeder Stelle im Code. Danach hat die Variable jedoch noch keinen Wert. Dies tut man über die *Initialisierung*. In PureBasic kann man alle drei Schritte gleichzeitig durchführen bzw. alle zwei, denn Deklaration und Definiton geschehen bei Variablen immer zusammen. *x = 1* deklariert und definiert also die Variable mit dem Bezeichner x und initialisiert sie mit dem Wert 1. Es ist zu beachten, dass Variablenbezeichner nicht mit Zahlen beginnen dürfen und keine Rechenoperatoren oder Sonderzeichen enthalten dürfen. Dazu zählen auch Umlaute und ähnliche Zeichen. *PrintN(Str(x))* gibt nun den Inhalt der Variable x, also 1, aus. Warum jedoch muss man schreiben *Str(x)*? Wenn man sich an Listing 1 erinnert, wurde dazu erklärt, dass *PrintN()* immer einen String als Argument erwartet, jedoch keine Zahl, wie z.B. *Delay()*. Deshalb muss x zuvor in einen String umgewandelt werden, was *Str()* erledigt. Es fällt auf, dass PureBasic innere Ausdrücke vor äußeren auswertet (*Str()* vor *PrintN()*), genauso wie man es in der Grundschulmathematik bei der Klammersetzung gelernt hat. Des Weiteren kann man Variablen einen neuen Wert zuweisen. Dies geschieht auf die gleiche Weise wie die Initialisierung. Außerdem kann mit Variablen gerechnet werden, was man ander Zeile *y = x+1* sehen kann. Die Variable repräsentiert hierbei den Wert, der ihr zuvor zugewiesen wurde, in diesem Fall 2.

In PureBasic sind alle Standardrechenmethoden verfügbar:

- '+' für Addition
- '-' für Substraktion
- '*' für Multiplikation
- '/' für Division

2.2.1 Konstanten

Wie man sah, können Variablen dynamisch Werte zugewiesen werden. Es gibt aber auch Fälle, in denen man eine Konstante haben will, also einen Bezeichner für einen Wert, der sich nicht ändert.

```
; Listing 4: Konstanten

#Vier = 4

PrintN(Str(#Vier))
Delay(2000)
```

```
Ausgabe:

4
```

In diesem Fall wurde die Konstante #Vier erzeugt (die Raute gehört zum Bezeichner!). Sie kann nachträglich nicht mehr geändert werden und repräsentiert in nachfolgenden Code den Zahlenwert 4. Ein nachträgliches *#Vier = 5* würde also einen Compilerfehler erzeugen.

Es ist zwar prinzipiell freigestellt, welche Bezeichner man für Konstanten und Variablen wählt, jedoch gilt die Konvention, dass Konstanten groß geschrieben werden und Variablen klein. Dies dient der besseren Unterscheidung und macht den Code insgesamt lesbarer.

PureBasic verfügt über viele interne Konstanten, z.B. *#Pi* für die Kreiszahl. Andere stehen in der PureBasic-Referenz, wobei die meisten zu diesem Zeitpunkt noch verwirren können, da sie für verschiedene Befehle als Optionen zur Verfügung stehen oder zur Verarbeitung dieser.

2.2.2 Eingabe

Natürlich möchte man auch die Möglichkeit haben, Variablenwerte vom Benutzer des Programmes festlegen zu lassen. Man denke dabei z.B. an einen Taschenrechner.

```
; Listing 5: Benutzereingabe

OpenConsole()

Print("1. Zahl: ")
x = Val(Input())
Print("2. Zahl: ")
y = Val(Input())

PrintN("x + y = "+Str(x+y))

Delay(2000)
```

```
Ausgabe:

1. Zahl: 1
2. Zahl: 3
x + y = 4
```

In diesem Beispiel wurden wieder viele neue Dinge eingeführt. Was sieht man bei der Ausführung? Das Programm gibt den ersten String *1. Zahl:* aus und wartet dann darauf das eine Zahl vom Benutzer eingegeben wird. Danach muss eine weitere Zahl eingegeben werden und zuletzt wird die Summe aus beiden ausgegeben.

Zuerst fällt auf, dass diesmal *Print()* und nicht *PrintN()* benutzt wird. Der Unterschied zwischen beiden liegt einzig und allein darin, dass *Print()* keinen Zeilenumbruch erzeugt, also die nächste Ausgabe auf die gleiche Zeile ausgegeben wird. Als nächstes ist der Befehl *Input()* neu. Er wartet bis der Benutzer etwas eingibt und Enter drückt. Die Eingabe wird als String zurückgegeben, das bedeutet man erhält von dem Befehl etwas (in diesem Fall einen String), mit dem man weiterarbeiten kann. Außerdem erzeugt *Input()* einen Zeilenumbruch nach der Eingabe. Da wir jedoch einen Zahlenwert speichern wollen und keinen String, muss dieser nun in einen Zahlenwert umgewandelt werden. Dies geschieht mit dem Befehl *Val()*, der als Gegenstück zu *Str()* gesehen werden kann. Es ist natürlich auch möglich einfach nur Enter zu drücken, wenn eine Eingabe erwartet wird, sodass ein leerer String (also "")

zurückgegeben wird. Zuletzt fällt auf, dass die Zahlenoperation, das Addieren, direkt in den *Str()*-Befehl geschrieben wurde. Es wird also zuerst der Ausdruck in den Klammer ausgewertet, bevor der Befehl an sich ausgewertet wird. Dies gilt auch für alle anderen Befehle. Des Weiteren kann man Strings zusammenfügen, ebenfalls über +.

2.3 Variablentypen

Es existieren mehrere sogenannter Variablentypen. Jede Variable hat einen ihr zugewiesenen Typ, wobei der voreingestelle *Integer* ist, was bedeutet, dass die Variable eine Ganzzahl ist und im Speicher 4 Byte bzw. 8 Byte einnimmt, je nachdem ob man ein 32- oder 64-Bit-System hat. Die Größe im Arbeitsspeicher ist entscheidend für die maximale Größe des Wertes.

Im Arbeitsspeicher sind alle Zahlen im binären System abgespeichert, d.h. als Nullen und Einsen, die Sprache des Computers. Ein Byte besteht aus 8 Bits, also 8 Stellen im binären System. Wenn nun ein Integer 4 Byte groß ist, können in ihm Zahlen abgespeichert werden, die im binären System insgesamt 32 Bits groß sind.

Da in PureBasic Variablen bis auf wenige Ausnahmen ein Vorzeichen erhalten, halbiert sich der Maximalwert einer Variable jedoch, da das Vorzeichen ebenfalls über das Binärsystem mit den Bits einer Variable repräsentiert wird.

Wenn man einer Variable den Maximalwert zuweist und dann noch eine 1 hinzuaddiert, beginnt die Zählung sozusagen von vorne, dass heißt wenn man auf einem 64-Bit-System einem Integer den Wert 9223372036854775807 zuweist und dann 1 hinzuaddiert, hat dieser Integer danach den Wert -9223372036854775808.

Es gibt andere Variablentypen neben Integer. Nachfolgend sind einige aufgeführt:

- Long, für Ganzzahlen im Bereich -2147483648 bis +2147483647 (4 Byte)
- Float, für Gleitkommazahlen (unbegrenze Größe)
- Quad, für große Ganzzahlen von -9223372036854775808 bis +9223372036854775807 (8 Byte)
- Ascii bzw. Unicode, für ein Zeichen (1 bzw. 2 Byte)
- String für Zeichenketten (Länge des Strings + 1 Byte)
- Andere Variablentypen stehen in der Referenz

Im Quellcode kann festgelegt werden, von welchem Typ eine Variable ist.

```
; Listing 6: Variablentypen

OpenConsole()

x.l = 5
y.f = 2
z.s = "Ich bin der String z"
```

```
PrintN("Ich bin der Long x: "+Str(x))
PrintN("Ich bin der Float y: "+StrF(y))
PrintN(z)
```

```
Delay(2000)
```

```
Ausgabe:

Ich bin der Long x: 5
Ich bin der Float y: 2.0000000000
Ich bin der String z
```

Man sieht, dass es ganz einfach ist Variablen einen Typ zuzuweisen: Man muss nur hinter den Bezeichner einen Punkt gefolgt von dem ersten Buchstaben des Variablentyps in Kleinschreibung schreiben. Wichtig ist außerdem zu sehen, dass bei der Umwandlung des Floats y in einen String der Befehl *StrF()* benutzt wird. Das hat mit der Formatierung der Ausgabe zu tun, *Str()* würde die Nachkommastellen abschneiden. Wenn also eine Gleitkommazahl anstatt einer Ganzzahl in einen String umgewandelt werden soll, benutzt man *StrF()*.

Die letzte Auffälligkeit ist die Stringvariable. Diese wird über ein kleines s definiert. Ihr kann, wie bei Zahlvariablen auch, ein Wert zugewiesen werden, dieser muss aber natürlich in Anführungstrichen stehen, da es sich um einen String handeln muss. Dementsprechend kann die Variable einfach so an *PrintN()* als Argument übergeben werden, da der Befehl einen String erwartet.

2.3.1 Typumwandlung

An mehreren Stellen wurden schon Typumwandlungen unternommen, z.B. wenn eine Zahlvariable als String ausgegeben wurde. Grundsätzlich ist es möglich jeden Variablenwert in einen anderen Typ umzuwandeln.

```
; Listing 7: Typumwandlung

OpenConsole()

x.l = 5
y.f = x
PrintN(StrF(y))

y = ValF(Input())
PrintN(StrF(y))

Delay(2000)
```

```
Ausgabe:

5.0000000000
2.5
2.5000000000
```

In der sechsten Zeile sieht man, wie ein Longwert einfach in eine Floatvariable gespeichert wird. Andersherum würde es auch funktionieren, die Nachkommastelle würde jedoch abgeschnitten werden. In der neunten Zeile wird der String, der von *Input()* zurückgegeben wird, in einen Float von *ValF()* umgewandelt wird. Man beachte, dass bei der Eingabe des Floats die amerikanische Konvention gilt, dass also ein Punkt anstatt eines Kommas geschrieben

wird. Bei der Benutzung von *Val()* gilt die gleiche Regel, wie bei der Benutzung von *Str()*. Wenn man einen String in eine Ganzzahl umwandeln will, benutzt man *Val()*, ansonsten *ValF()*.

Val()- und *Str()*-Befehle werden also bei der Umwandlung von Zahlen in Strings und anders herum benutzt. Zahlvariablen können untereinander ohne Befehle in Variablen anderen Typs gepeichert werden.

2.3.2 Strings

Es wurden jetzt schon an mehreren Stellen Strings benutzt. Wie diese grundsätzlich funktionieren, sollte inzwischen klar geworden sein. Es gibt jedoch noch einige andere nützliche Funktionen.

```
; Listing 8: Strings

OpenConsole()

string.s = "Ich bin ein String!"
PrintN(Str(CountString(string, "String")))

PrintN(LCase(string))

string = InsertString(string, "neuer ",13)
PrintN(string)

string = RTrim(string, "!")
PrintN(string)
PrintN(Left(string,3))
```

```
Ausgabe:

1
ich bin ein string!
Ich bin ein neuer String!
Ich bin ein neuer String
Ich
```

Nachfolgend sind einige Befehle zum Arbeiten mit Strings aufgeführt.

- *CountString()* zählt wie oft der angegebene String im Übergebenen vorkommt.
- *LCase()* wandelt alle Großbuchstaben in Kleinbuchstaben um und gibt das Ergebnis zurück.
- *InsertString()* fügt einen neuen String an der angegeben Stelle ein und gibt das Ergebnis ebenfalls zurück.
- *RTrim()* entfernt den angegeben String von Rechts.
- *Left()* gibt soviele Zeichen ab dem linken Rand zurück, wie angegeben wurde.

Weitere nützliche Befehle für Strings stehen im Referenz-Handbuch.

2.3.3 Aufgaben

1. Es soll ein Programm geschrieben werden, bei dem der Benutzer einen Kreisradius angibt, worauf der Umfang des Kreises angegeben wird. Die Formel für den Umfang lautet: 2 x Radius x Pi
2. Es soll ein Programm geschrieben werden, dass den Sinus eines Winkels ausrechnet. Die Befehle hierfür wurden noch nicht vorgestellt, deshalb müssen sie in der Referenz nachgeschlagen werden. Die nötigen Befehle befinden sich in der Bibliothek Math.

2.4 Bedingungen

Bisher waren die kleinen Beispielprogramme sehr undynamisch, soll heißen, sie haben von oben nach unten die Befehle abgearbeitet. Es gibt jedoch sogenannte Kontrollstrukturen, mit denen man den Ablauf dynamischer gestalten kann.

```
; Listing 9: If-Else-Klausel

OpenConsole()

x = Val(Input())

If x < 3
  PrintN("x ist kleiner als 3")
ElseIf x = 4
  PrintN("x ist genau 4")
Else
  PrintN("x ist größer als 4")
EndIf

Delay(2000)
```

```
Ausgabe:

2
x ist kleiner als 3

4
x ist genau 4

5
x ist größer als 4
```

Wenn man das Programm kompiliert und ausführt, erwartet es die Eingabe einer ganzen Zahl. Je nachdem, ob man nun einer Zahl kleiner 3, genau 4 oder eine andere eintippt, erhält man eine andere Ausgabe. Man erhält dieser Verhalten durch sogenannte If-Else-Klauseln, die nach logischen Prinzipien auf die Werte reagieren. Die Schlüsselwörter (Also Wörter, die die Sprache an sich ausmachen und keine Befehle, Konstanten oder Variablen sind, sondern zu ihrer Realisierung vonnöten sind) sind dabei wörtlich aus dem Englischen zu übersetzen: Wenn (*If*) x kleiner als 3 ist..., ansonten, wenn (*ElseIf*) x genau 4 ist..., ansonten (*Else*)...

Das "kleiner als"-Zeichen ($<$) und das "ist gleich"-Zeichen ($=$) sind sogenannte Vergleichsoperatoren, d.h. sie vergleichen zwei Werte. Werte können hierbei vieles sein: Variablen, Konstanten, Strings, Zahlenwerte. Es gibt natürlich auch ein "größer als"-Zeichen ($>$), ein

"ist ungleich"-Zeichen (<>) und ein "größer oder gleich"- bzw. "kleiner oder gleich"-Zeichen (>= bzw. <=).

Wichtig ist zu verstehen, dass die Vergleiche von oben nach unten durchgeführt werden, wenn jedoch ein Vergleich zutrifft (man sagt er ist *wahr*, ansonsten ist er *falsch*), z.B. im ersten Fall x wirklich kleiner als 3 ist, werden die Befehle unter dem Vergleich durchgeführt und die anderen Vergleiche ignoriert, d.h. das Programm springt sofort in der Programmausführung unter *EndIf*.

2.4.1 Logische Verknüpfungen

If-Else-Klauseln helfen also dabei, ein Programm dynamischer zu gestalten. Trotzdem sind mit diesen noch nicht alle Fälle abgedeckt, die bei Vergleichen auftreten können. Was ist z.B. wenn zwei Bedingungen gleichzeitig erfüllt sein sollen? Hier kommen logische Verknüfungen zum Tragen.

```
; Listing 10: And, Or

OpenConsole()

x = Val(Input())
y = Val(Input())

If x <= 3 And y >= 2
  PrintN("x ist kleiner oder gleich 3 und y ist größer oder gleich 2")
ElseIf x > 3 Or y < 2
  PrintN("x ist größer als 3 oder y ist kleiner als 2")
EndIf

Delay(2000)
```

```
Ausgabe:

3
2
x ist kleiner oder gleich 3 und y ist größer oder gleich 2

5
2
x ist größer als 3 oder y ist kleiner als 2
```

Das Programm funktioniert wie "Listing 9", jedoch werden diesmal 2 Eingaben erwartet. Neben den neuen Vergleichsoperatoren, die auch schon oben erwähnt wurden, werden hier die logischen Verknüpfungen eingeführt. Auch hier kann wieder wörtlich übersetzt werden. Im ersten Fall muss x kleiner oder gleich 3 sein *und* y muss größer oder gleich 2 sein. Falls der erste Fall jedoch nicht zutrifft, reicht es im zweiten Fall, wenn x größer als 3 ist *oder* y kleiner als zwei ist. Solche Verknüpfungen können beliebig komplex sein. Außerdem ist es möglich nach mathematischen Regeln Vergleiche einzuklammern, sodass bestimmte Vergleiche vor anderen getätigt werden und deren Ergebniss zurückgegeben wird, z.B. *(... Or ...) And ...*. Es muss also der eingeklammerte Vergleich zutreffen und ein weiterer, ansonsten ist die gesamte Bedingung *falsch*.

Es gibt noch weitere logische Verknüpfungen, wie z.B. das sogenannte *Exklusiv-Oder*. Wie diese benutzt werden, steht in der Referenz.

Eine Sonderrolle nimmt die logische Verknüpfung *Not* ein, die einen Vergleich umkehrt. Eine If-Klausel würde dann ausgeführt werden, wenn ein Vergleich falsch wäre. *If Not* $x < 3$ wäre also das gleiche wie *If* $x >= 3$.

2.4.2 Fallunterscheidung mit Select

Eine weitere Möglichkeit ein Programm dynamischer zu gestalten, ist die sogenannte Fallunterscheidung. Dafür gibt es das Select-Schlüsselwort.

```
; Listing 11: Select

OpenConsole()

x = Val(Input())

Select x
  Case 1
    PrintN("x ist gleich 1")
  Case 2, 3, 4
    PrintN("x ist gleich 2, 3 oder 4")
  Case 5 To 10, 11
    PrintN("x hat einen Wert zwischen 5 und 10 oder ist 11")
  Default
    PrintN("x ist größer als 11 oder kleiner als 1")
EndSelect

Delay(2000)
```

```
Ausgabe:

1
x ist gleich 1

4
x ist gleich 2, 3 oder 4

-1
x ist größer als 11 oder kleiner als 1

9
x hat einen Wert zwischen 5 und 10 oder ist 11
```

Wieder wird die Eingabe von x erwartet. Dann wird der Wert von x unterscheidet. Es werden immer die Befehle ausgeführt, die unter dem eingetroffenen Fall stehen. Wenn x also 1 ist, wird der Befehl *PrintN("x ist gleich 1")* ausgeführt und danach springt das Programm direkt zu *Delay(2000)*. Außerdem ist es möglich mehrere Fallunterscheidungen zu kombinieren, damit nicht jeder Fall einzeln betrachtet werden muss, wenn die gleichen Befehle ausgeführt werden sollen. Dafür schreibt man entweder die Fälle mit Kommata getrennt oder benutzt das *To*-Schlüsselwort, das nur für Zahlen funktioniert und alle Zahlen von ... bis (To) ... unterscheidet. Wenn kein Fall zutrifft, wird der optionale Standardfall (Default) ausgeführt.

Zuletzt sei gesagt, dass neben Strings nur ganze Zahlen unterschieden werden. Wenn man einen Float übergibt, wird die Zahl zur nächsten Ganzzahl abgerundet.

2.4.3 Aufgaben

1. Es soll das Umfangsprogamm aus dem vorigen Kapitel so abgeändert werden, dass es einen Umfang von 0 ausgibt, wenn ein negativer Radius eingegeben wird.
2. Es soll ein Programm geschrieben werden, bei dem man die Geschwindigkeit eines Autos eingibt und das darauf ausgibt, ob der Wagen zu schnell fährt und wenn, wieviel zu schnell. Als Tempolimit kann 50 km/h angenommen werden.

2.5 Schleifen

Es gibt Fälle, in denen man einen Programmteil mehrmals hintereinander wiederholen will. Dafür gibt es weitere Kontrollstrukturen, sogenannte Schleifen.

2.5.1 For-Schleife

Die For-Schleife führt einen Block von Befehlen eine festgelegte Anzahl aus.

```
; Listing 12: For-Schleife

OpenConsole()

For x = 1 To 10
  PrintN(Str(x))
Next

Delay(2000)
```

```
Ausgabe:

1
2
3
4
5
6
7
8
9
10
```

Die Ausgabe erzeugt die Zahlen 1 bis 10 untereinander geschrieben. Man übergibt *For* eine Variable, die direkt hinter *For* einen Wert zugewiesen bekommen muss. Dann schreibt man bis zu welchem Wert die Variable hochgezählt werden soll. Es wird jeder Befehl bis *Next* ausgeführt, dann wird zu x eine 1 hinzuaddiert und der Block an Befehlen wird erneut ausgeführt. Eine Ausführung des gesamten Blocks ist eine sogenannte *Iteration*. Sobald x größer als 10 ist, geht die Ausführung unter *Next* weiter.

Es ist auch möglich, die Zahl, die mit jeder Iteration zu x hinzuaddiert wird, zu modifizieren.

```
; Listing 13: Step

OpenConsole()

For x = 1 To 10 Step 2
  PrintN(Str(x))
Next

Delay(2000)
```

```
Ausgabe:

1
3
5
7
9
```

Über das *Step*-Schlüsselwort wird festgelegt, dass in diesem Fall immer 2 anstatt 1 hinzuaddiert wird. *Step* kann auch negativ sein.

```
; Listing 14: Negativer Step

OpenConsole()

For x = 10 To 1 Step -1
  PrintN(Str(x))
Next

Delay(2000)
```

```
Ausgabe:

10
9
8
7
6
5
4
3
2
1
```

2.5.2 While und Repeat

Neben der For-Schleife gibt es außerdem die While- und die Repeat-Schleife.

```
; Listing 15: While

OpenConsole()

While x <= 10
  PrintN(Str(x))
  x+1
Wend
```

```
Delay(2000)
```

```
Ausgabe:

0
1
2
3
4
5
6
7
8
9
10
```

Der While-Schleife muss ein Vergleich übergeben werden und die Schleife wird solange ausgeführt, wie dieser zutrifft. D.h. es gibt auch Fälle, in denen die Schleife, im Gegensatz zur For-Schleife, garnicht ausgeführt wird, nämlich wenn x schon vor der Ausführung der Schleife größer als 10 wäre. Außerdem sieht man an diesem Beispiel, dass eine Variable ohne Initialisierung den Wert 0 hat. In vielen anderen Programmiersprachen wäre dies schlechter Stil oder sogar ein Fehler, in PureBasic ist dies gewollt.

```
; Listing 16: Repeat

OpenConsole()

Repeat
  PrintN(Str(x))
  x+1
Until x >= 10

Delay(2000)
```

```
Ausgabe:

0
1
2
3
4
5
6
7
8
9
```

Die Repeat-Schleife unterscheidet sich insofern von der While-Schleife, als dass die Schleife immer mindestens einmal ausgeführt wird und außerdem solange ausgeführt wird *bis* ein bestimmer Fall eintritt und nicht *solange* einer zutrifft.

Es ist möglich jede Schleife als eine While-Schleife zu schreiben.

2.5.3 Break und Continue

Bisher haben die Schleifen einfach strikt den Befehlsblock abgearbeitet. Es ist aber möglich, diese Ausführung dynamischer zu gestalten.

```
; Listing 17: Break und Continue

OpenConsole()

For x = 1 To 10
  If x = 5
    Continue
  ElseIf x = 8
    Break
  EndIf

  PrintN(Str(x))
Next

Delay(2000)
```

```
Ausgabe:

1
2
3
4
6
7
```

Man sieht das nur die Zahlen von 1 bis 7 ausgegeben werden, ohne die 5 und auch ohne die Zahlen 8 bis 10, obwohl man es von der For-Anweisung erwarten würde. Dies liegt an den Schlüsselwörtern *Continue* und *Break*. *Continue* springt sofort zur nächsten Iteration, ohne dass der restliche Befehlsblock ausgeführt wird. *Break* bricht die gesamte Schleife sofort ab, ohne das die nachfolgenden Iterationen noch beachtet werden.

2.5.4 Aufgaben

1. Es soll ein Programm geschrieben werden, dass nach einem String und einer ganzen positiven Zahl n fragt und den String n-mal ausgibt.
2. Es soll ein Programm geschrieben werden, das den Benutzer immer wieder zu einer Eingabe auffordert und den String ausgibt, bis er einfach nur Enter drückt.

2.6 Arrays, Listen und Maps

Es gibt Fälle, in denen man mehrere zusammenhängende Werte speichern möchte. Nun könnte man für jeden Wert eine eigene Variable anlegen, was jedoch unvorteilhaft wäre und zu Spaghetti-Code führen kann, also zu sehr schwer leslichen und unübersichtlichen Code. Dafür gibt es Arrays.

```
; Listing 18: Arrays

OpenConsole()

Dim array.l(2)

For x = 0 To 2
  array(x) = x
Next

For x = 0 To 2
  PrintN(Str(array(x)))
Next

Delay(2000)
```

```
Ausgabe:

0
1
2
```

In der 5. Zeile wird das Array deklariert und definiert. Dazu gibt es das Schlüsselwort *Dim*. Man schreibt dahinter den Bezeichner für das Array und in runden Klammern, die zum Bezeichner gehören, den höchsten sogenannten *Index*. Der Index wird benötigt, um auf die einzelnen Variablen, die im Array gespeichert sind, zuzugreifen. Der niedrigste Index ist 0. Die Typzuweisung soll verdeutlichen, wie man Arrays unterschiedlichen Typs erzeugt, nämlich genauso, wie bei normalen Variablen auch. Ein Array nimmt immer die Anzahl der Elemente mal die Größe einer einzelnen Variable des Arrays im Speicher ein, in diesem Fall also 12 Byte, da es sich um ein Array von Longs handelt (3 x 4 Byte). In der ersten For-Schleife sieht man, wie ein Zugriff auf ein Array aussieht: Man schreibt in die Klammern des Bezeichners den Index auf den man zugreifen will. Durch die For-Schleife greift man nacheinander auf alle Indizes zu. Genauso kann man auch die Werte der Variablen im Array abrufen, was in der zweiten For-Schleife geschieht.

2.6.1 Größenänderung

Es ist möglich die Größe eines Arrays im Programm zu ändern.

```
; Listing 19: ReDim

OpenConsole()

Dim a.l(2)

For x = 0 To 2
  array(x) = x
Next

For x = 0 To 2
  PrintN(Str(a(x)))
Next

ReDim a(3)
```

```
a(3) = 3
PrintN(Str(a(3)))

Delay(2000)
```

```
Ausgabe:

0
1
2
3
```

Das Beispiel unterscheidet sich insofern von dem ersten, als dass weiter unten das Array um ein Element erweitert wird, nämlich über das Schlüsselwort *ReDim*. Man schreibt dabei in die Klammern des Bezeichners die neue Größe des Arrays. Wenn die neue Größe kleiner ausfällt als die vorige wird der Rest "abgeschnitten".

2.6.2 Mehrdimensionale Arrays

Mehrdimensionale Arrays kann man sich als Arrays von Arrays vorstellen. Unter jedem Index des ersten Arrays ist ein weiteres Array. Bildlich kann man sich das so vorstellen: Ein einfaches Array ist eine Reihe von Variablen, ein zweidimensionales Array ist dann ein Schachbrett-ähnliches Gebilde und ein dreidimensionales ein Raum. Ab vier Dimensionen hinkt der Vergleich jedoch, dann sollte man sich der Baumdarstellung als Array von Arrays von Arrays... bemühen.

```
; Listing 20: Arrays

OpenConsole()

Dim a.l(2,2)

For x = 0 To 2
  For y = 0 To 2
    a(x,y) = x
  Next
Next

For x = 0 To 2
  For y = 0 To 2
    Print(Str(a(x,y)))
  Next
PrintN("")
Next

Delay(2000)
```

```
Ausgabe:

000
111
222
```

Ein mehrdimensionales Array wird erzeugt, indem man in die Klammern mehrere maximale Indizes getrennt durch Kommata schreibt. In diesem Fall wurde z.B. ein zweidimensionales

Array erzeugt. Wenn man ein dreidimensionales erzeugen möchte, müsste ein weiterer maximaler Index, wieder durch ein Komma getrennt, hinzugefügt werden. Außerdem wurde hier die sogenannte *Verschachtelung* angewandt. Es wurde eine For-Schleife in eine For-Schleife geschrieben, d.h. es wird in der ersten Iteration der äußeren Schleife die komplette innere ausgewertet. In der zweiten wird wieder die komplette innere ausgewertet usw. Dies macht man sich zunutze, um das mehrdimensionale Array zu füllen: Der erste Index bleibt gleich und es wird das komplette darunterliegende Array gefüllt. Dann wird das nächste gefüllt usw.

Man beachte, dass bei mehrdimensionalen Arrays nur die Größe der letzten Dimension geändert werden kann. Im Fall aus *Listing 20* müsste man schreiben *ReDim a(2,neue_größe)*.

2.6.3 Linked Lists

Linked Lists (Listen) sind dynamische Datenstrukturen, die sich dadurch abheben, dass dynamisch Elemente hinzugefügt und entfernt werden können. Im Prinzip sind es also dynamische Arrays, auch wenn die interne Darstellung eine ganz andere ist.

```
; Listing 21: Linked Lists

OpenConsole()

NewList l.l()

For x = 1 To 10
  AddElement(l())
  l() = 1
Next

SelectElement(l(),2)
InsertElement(l())

l() = 15

FirstElement(l())
For x = 1 To 11
  PrintN(Str(l()))
  NextElement(l())
Next

Delay(2000)
```

```
Ausgabe:

1
1
15
1
1
1
1
1
1
1
1
```

Es werden untereinander 11 Zahlen ausgegeben, nämlich zehn Einsen, sowie an der dritten Stelle eine 15.

Zuerst wird eine neue Liste mittels *NewList* erzeugt, wobei die Klammern wie bei Arrays zum Bezeichner gehören. Mit *AddElement()* wird immer wieder ein neues Element zur Liste hinzugefügt, das danach einen Wert zugewiesen bekommt. Man sieht das kein Index von Nöten ist, vielmehr verwaltet PureBasic für jede Liste einen internen Index und man muss selber mit den verfügbaren Befehlen dafür sorgen, dass auf das richtige Element zugegriffen wird. *AddElement()* setzt immer den Index auf das neue Element. Mit *SelectElement()* kann man den Index der Liste ändern, wobei wie bei Arrays das erste Element den Index 0 hat. *InsertElement()* fügt ein neues Element vor dem aktuellen ein und setzt den Index auf dieses. *FirstElement()* setzt den Index zurück auf das erste Element, sodass anschließend mit der For-Schleife die Liste ausgegeben werden kann, wobei *NextElement()* den Index auf das nächste Element setzt.

Weitere Befehle stehen in der Handbuch-Referenz.

2.6.4 Maps

Maps sind Datenstrukturen, in denen die Elemente nicht über Indizes referenziert werden, sondern über *Schlüssel*.

```
; Listing 22: Maps

OpenConsole()

NewMap initialen.s()

initialen("HP") = "Hans Peter"
initialen("KS") = "Kirill Schuschkow"

PrintN(initialen("HP"))
PrintN(initialen("KS"))

Delay(2000)
```

```
Ausgabe:

Hans Peter
Kirill Schuschkow
```

Maps werden ähnlich wie Listen angelegt, durch das Schlüsselwort *NewMaps*. Jedoch muss man neue Element nicht über einen Add-Befehl hinzufügen, sondern es reicht auf die Map mit dem neuen Schlüssel zuzugreifen und gleichzeitig mit dem Wert für diesen Schlüssel zu initialisieren; das Element wird dann mit dem Wert neu angelegt. Über diese Schlüssel kann dann auf die Elemente zugegriffen werden. Die PureBasic-Referenz stellt vielseitige Befehle zu Maps vor.

2.6.5 ForEach-Schleife

Mit der ForEach-Schleife ist es möglich alle Elemente von Listen und Maps einfach zu durchlaufen.

```
; Listing 23: ForEach-Schleife

OpenConsole()

NewMap initialen.s()

initialen("HP") = "Hans Peter"
initialen("KS") = "Kirill Schuschkow"

ForEach initialen()
  PrintN(initialen())
Next

Delay(2000)
```

```
Ausgabe:

Hans Peter
Kirill Schuschkow
```

Die ForEach-Schleife duchläuft alle Elemente einer Map, ohne dass man die Schlüssel eingeben muss. *initialen()* repräsentiert dann immer das aktuelle Element, das dementsprechend über diesen Bezeichner verändert werden kann.

Die ForEach-Schleife funktioniert auf die gleiche Art und Weise mit Linked Lists.

2.6.6 Aufgaben

1. Es soll ein Programm geschrieben werden, dass ein Menü anzeigt. Es soll die Möglichkeit geben ein zweidimensionales Feld auszugeben, das standardmäßig mit Nullen gefüllt ist, oder eine Koordinate einzugeben. Wenn eine Koordinate eingegeben wird, soll auf das Feldelement, das durch die Koordinate repräsentiert wird, eine 1 addiert werden.

3 Fortgeschrittene Themen

3.1 Strukturen

Strukturen sind dazu da, um Variablen sinnvoll zu ordnen. Man kann mit ihnen eigene Datentypen definieren und so alle Daten eines einzelnen Bestandes zusammenfassen.

```
; Listing 24: Strukturen, With

Structure Person
  name.s
  alter.l
EndStructure

OpenConsole()

Dim alle.Person(2)

For x = 0 To 2
  With alle(x)
    \alter = x
    \name = "Hans"
  EndWith
Next

For x = 0 To 2
  PrintN("Alter: "+Str(alle(x)\alter))
  PrintN("Name: "+alle(x)\name)
Next

Delay(2000)
```

```
Ausgabe:

Alter: 0
Name: Hans
Alter: 1
Name: Hans
Alter: 2
Name: Hans
```

Zu Anfang wird der Strukturtyp im *Structure : EndStructure*-Block definiert. Der neu erzeugte Typ erhält den Namen *Person* und enthält die Stringvariable "name" und die Longvariable "alter" (man nennt diese Variablen *Strukturvariablen*). Dann wird ein Array von diesem neuen Typ erzeugt und gefüllt. Dazu ist grundsätzlich nicht das Schlüsselwort *With* vonnöten. *With* wird eine definierte Struktur übergeben und in diesem Block wird über einen Backslash und den Strukturvariablennamen auf die Strukturvariable zugegriffen. Wie man in der zweiten For-Schleife sieht, kann auch ohne *With* auf eine Strukturvariable zugegriffen werden, indem man vor den Backslash noch die definierte Struktur schreibt.

3.2 Prozeduren

Prozeduren sind ein Mittel, um immer wieder benötigten Code einfach zur Verfügung zustellen. Anstatt den gleichen Code immer und immer wieder in die Quelldatei zu schreiben, definiert man eine Prozedur. Prozeduren sind im Grunde nichts anderes, als die schon vorhandenen Befehle von PureBasic, *PrintN()* ist z.B. eine.

```
; Listing 25: Prozeduren

Procedure.l square(x.l)
  ProcedureReturn x*x
EndProcedure

OpenConsole()

PrintN(Str(square(2)))

Delay(2000)
```

```
Ausgabe:

4
```

Dieses einfache Beispiel verdeutlicht, wozu Prozeduren in der Lage sind: Anstatt immer wieder *x*x* einzutippen, schreibt man *square(x)*, wodurch der Quellcode insgesamt lesbarer wird, da immer sofort erkennbar ist, was gemeint wurde. Natürlich kann man auch weitaus komplexeren Code in eine Prozedur schreiben.

Hinter dem Schlüsselwort *Procedure* wurde ein Typ übergeben (Long). Dieser ist wichtig, da er definiert, was für einen Typ der Rückgabewert hat. Der Rückgabewert ist das, was hinter *ProcedureReturn* steht. Nach der Auswertung der Prozedur erhält *Str()* diesen Wert und kann ihn weiterverarbeiten, genauso wie *Str()* einen String als Rückgabewert hat, der von *PrintN()* weiter verarbeitet werden kann. Nachdem *ProcedureReturn* aufgerufen wurde, springt die Programmausführung aus der Prozedur wieder zu der Stelle, an der die Prozedur aufgerufen wurde.

In den Klammern des Prozedurenbezeichners ist ein sogenanntes Argument. Dieses erhält einen Namen und einen Typ. Beim Aufruf der Prozedur muss immer ein Long als Argument übergeben werden, genauso wie man *Str()* einen Long übergeben muss, damit dieser in einen String umgewandelt werden kann. Eine Prozedur kann auch mehrere Argumente erhalten. Es ist zu beachten, dass nicht die Variable an sich übergeben wird, sondern der Wert der Variable in eine interne Prozedurvariable mit dem Argumentennamen als Bezeichner kopiert wird. Der Rückgabetyp, der Bezeichner und die Argumente ergeben zusammen den sogenannten *Prozedurkopf*.

In anderen Programmiersprachen wird das, was in PureBasic Prozeduren sind, normalerweise Funktionen genannt. Eigentlich haben Prozeduren keinen Rückgabewert, Funktionen hingegen immer, das Konzept der Funktionen wurde jedoch erst nach den Prozeduren entwickelt.

3.2.1 Sichtbarkeit

Sichtbarkeit bedeutet, dass eine Variable, die außerhalb einer Prozedur definiert wurde, unter Umständen nicht in der Prozedur verfügbar ist.

```
; Listing 26: Sichtbarkeit

Procedure scope()
  x = 5
  PrintN(Str(x))
EndProcedure

OpenConsole()

x = 2
scope()
PrintN(Str(x))

Delay(2000)
```

```
Ausgabe:

5
2
```

Erst wird 5 ausgegeben und dann die 2, obwohl in der Prozedur $x = 5$ steht. Man könnte also annehmen, dass zweimal eine 5 augegeben werden müsste. Dies liegt daran, dass x in der Prozedur ein anderes ist, als das außerhalb. Es gibt Schlüsselwörter, wie *Global*, *Protected* und *Shared*, die dazu da sind, diesen Umstand zu umgehen, die Benutzung dieser ist jedoch in der modernen Programmierung verpönnt. Diese Schlüsselwörter stammen noch aus einer Zeit, als es noch keine Rückgabewerte und Argumente gab. Man arbeitet heutzutage über Zeiger, auf die in einem anderen Kapitel eingegangen wird.

3.2.2 Static

Das Schlüsselwort *Static* wird dazu benutzt, um eine Variable in einer Funktion einmal zu definieren und zu initialiseren, jedoch kein weiteres mal.

```
; Listing 27: Static

Procedure static_beispiel()
  Static  a = 1
  a+1
  Debug a
EndProcedure

OpenConsole()

static_beispiel()
static_beispiel()
static_beispiel()

Delay(2000)
```

```
Ausgabe:

2
3
4
```

Die Ausgabe ergibt hintereinander 2, 3 und 4, obwohl in der Prozedur *a = 1* steht. Dies liegt am Schlüsselwort *Static*, dass eine neue Definition und Initialisierung verhindert.

3.2.3 Arrays als Argument

Es ist auch möglich Arrays als Argument zu übergeben.

```
; Listing 28: Array als Argument

Procedure array_argument(Array a(1))
  For x = 2 To 0 Step -1
    a(x) = 2-x
  Next
EndProcedure

OpenConsole()

Dim a(2)
For x = 0 To 2
  a(x) = x
Next

array_argument(a())

For x = 0 To 2
  PrintN(Str(a(x)))
  Debug a(x)
Next

Delay(2000)
```

```
Ausgabe:

2
1
0
```

Wenn man ein Array als Argument übergibt, muss man in den Klammern des Arraybezeichners die Anzahl der Arraydimensionen angeben und vor den Bezeichner das Schlüsselwort *Array* schreiben. Wichtig ist, dass Arrays nicht wie normale Variablen kopiert werden, sondern als sogenannte Referenz übergeben werden, d.h. das Array in der Prozedur ist *dasselbe* Array wie außerhalb der Prozedur. Alle Änderungen die in der Prozedur am Array vorgenommen werden, sind auch außerhalb der Prozedur sichtbar, deswegen werden die Zahlen rückwärts ausgegen, weil sie in der Prozedur in dasselbe Array geschrieben werden. Deshalb muss man Arrays nicht als Rückgabewert zurückgeben, genauer: es geht garnicht.

Die gleichen Regeln gelten auch für Maps und Listen, man benutzt dann das Schlüsselwort *Map* bzw. *List*.

3.2.4 Declare

Es gibt Fälle, in denen man eine Prozedur aufrufen möchte, diese jedoch noch nicht definiert ist. Hier kommen Prozedurprototypen ins Spiel.

```
; Listing 29: Declare

OpenConsole()

Declare.l square(x.l)

PrintN(Str(square(2)))

Procedure.l square(x.l)
  ProcedureReturn x*x
EndProcedure

Delay(2000)
```

```
Ausgabe:

4
```

In diesem Fall wird die Prozedur *square()* aufgerufen, bevor sie definiert wurde. Deshalb deklariert man sie vorher mit dem Schlüsselwort *Declare*, wodurch dieses Verhalten erst möglich wird. Der Prozedurkopf muss dabei genau der gleiche sein, wie der bei der Definition der Prozedur. Dieses Verhalten ist vorallem dann hilfreich, wenn man eine Prozedur in einer anderen aufrufen möchte, obwohl sie erst unter dieser definiert wird.

3.2.5 Rekursion und Iteration

Es ist möglich eine Prozedur innerhalb einer anderen aufzurufen. Wenn eine Prozedur sich selber aufruft, spricht man von Rekursion. So ist es z.B. möglich die Fakultät einer Zahl zu berechnen.

```
; Listing 30: Rekursion

OpenConsole()

Procedure.l fak(n.l)
  If n <= 1
    ProcedureReturn 1
  Else
    ProcedureReturn n * fak(n-1)
  EndIf
EndProcedure

Print("Bitte geben Sie eine natürliche Zahl ein: ")
n = Val(Input())
PrintN("Die Fakultät lautet: "+Str(fak(n)))

Delay(2000)
```

```
Ausgabe:

Bitte geben Sie eine natürliche Zahl ein: 4
Die Fakultät lautet: 24
```

Es wird eine Prozedur *fak()* definiert, die die Fakultät rekursiv berechnen soll. Die Prozedur hat eine sogenannte *Abbruchbedingung*, d.h. in diesem Fall ruft sie sich nicht mehr selber auf. Gebe es diese Bedingung nicht, würde sich die Funktion immer wieder selber aufrufen, was zum Absturz des Computers führen würde, da der Arbeitsspeicher überfüllt wird. In allen anderen Fällen multipliziert sie die eingegebene Zahl mit allen Zahlen die niedriger als die Eingebene sind, bis zur 1, denn das wäre die Abbruchbedingung. Dies geschieht über Rekursion, die Prozedur ruft sich selber auf. Um genau zu verstehen was geschieht, sollte man einmal den Rückgabewert ausschreiben: *4 * fak(3)*. Am Anfang steht die eingegebene 4 und danach wird *fak(3)* aufgerufen. Nun ist n 3, es wird also aufgerufen *3 * fak(2)*. *fak(3)* steht also für *3 * fak(2)*, man erhält *4 * 3 * fak(2)*. *fak(2)* gibt wiederum *2 * fak(1)* zurück, man hat bisher als Rückgabewert also *4 * 3 * 2 * fak(1)*. *fak(1)* erfüllt die Abbruchbedingung und gibt einfach 1 zurück. Der entgültige Rückgabewert lautet also *4 * 3 * 2 * 1*, was die Fakultät von 4 (24) ist.

Ein weiteres Verfahren wäre die Iteration, ein Verfahren, das in PureBasic über Schleifen genutzt wird.

```
; Listing 31: Iteration

OpenConsole()

Procedure.l fak(n.l)
  fak = 1
  For k = 1 To n
    fak * n
  Next
  ProcedureReturn fak
EndProcedure

Print("Bitte geben Sie eine natürliche Zahl ein: ")
n = Val(Input())
PrintN("Die Fakultät lautet: "+Str(fak(n)))

Delay(2000)
```

```
Ausgabe:

Bitte geben Sie eine natürliche Zahl ein: 4
Die Fakultät lautet: 24
```

Das Beispiel bedarf keiner weiteren Erläuerung, wenn man Schleifen verstanden hat. Iterative Verfahren sind meistens weitaus schneller als rekursive und können außerdem den Arbeitsspeicher nicht zum Absturz bringen. Nach Möglichkeit sollte man also immer nach iterativen Verfahren suchen.

3.2.6 Aufgaben

1. Es soll ein Programm geschrieben werden, in dem einer Prozedur ein Array von Zahlen übergeben wird und die alle Zahlen ausgibt, die größer als der Durchschnitt sind. Der Durchschnitt berechnet sich über die Gesamtsumme der Zahlen geteilt durch die Anzahl der Zahlen. Der Inhalt des Arrays kann fest einprogrammiert werden und muss nicht unbedingt vom Benutzer eingegeben werden.
2. Es soll ein Programm geschrieben werden, das die Fibonacci-Folge bis zu einer eingebenen Stelle berechnet. Diese soll dann augegeben werden. Eine Stelle der Fibonacci-Folge berechnet sich als die Summe der beiden vorherigen. Die nullte Stelle ist 0 und die erste 1.

3.3 Code-Auslagerung

Sobald Projekte eine gewisse Größe erreichen, wird es sinnvoll den Code auf mehrere Dateien aufzuteilen.

ausgelagert.pb

```
Procedure ausgelagert()
  PrintN("Ich bin eine ausgelagerte Prozedur")
EndProcedure
```

main.pb

```
; Listing 32: Code-Auslagerung

IncludeFile "./ausgelagert.pb"

OpenConsole()

ausgelagert()

Delay(2000)
```

```
Ausgabe:

Ich bin eine ausgelagerte Prozedur
```

Nachdem man den Code in die zwei Dateien eingetippt und im gleichen Ordner abgespeichert hat, kann die Prozedur *ausgelagert()* in der Hauptdatei aufgerufen werden. Dies liegt am Schlüsselwort *IncludeFile*, dass den Inhalt der übergebenen Datei an genau dieser Stelle einfügt. Als Ausdruck wird ein String benötigt, der den Pfad zur ausgelagerten Datei enthält. Theoretisch kann man auch normale Befehle, Variablen, Arrays etc. auf diese Weise auslagern, normalerweise lagert man aber Prozeduren aus, die aus der Hauptdatei aufgerufen werden sollen.

Mit dem Schlüsselwort *XIncludeFile* wird das Gleiche erreicht, jedoch verhindert dieses im Gegensatz zum anderen, dass eine Datei mehrmals eingefügt wird.

3.3.1 Bibliotheken

Die obere Variante bietet sich bei Software-Projekten an, bei welchen der ausgelagerte Code speziell für dieses Projekt entwickelt wurde. Wenn man jedoch Prozeduren schreibt, die von vielen Programmen benutzt werden können, erstellt man normalerweise eine Bibliothek. Ein Beispiel für solch eine Bibliothek ist z.B. die Befehlsbibliothek mit mathematischen Befehlen für PureBasic.

Bibliotheken haben einige Vorteile gegenüber der ersten Variante. So wird der Bibliothekscode, auch wenn er von mehreren Programmen gleichzeitig benutzt wird, nur einmal geladen, die Ressourcen des Computers werden also effektiver genutzt. Außerdem ist der Speicherplatz einer gespeicherten Bibliothek geringer, da diese auf dem Computer in kompilierter Form vorliegen. Unter Windows sind dies die .dll-Dateien, unter Linux die .so-Dateien und unter Mac OS X die .dylib-Dateien.

In eine Bibliotheksdatei schreibt man normalerweise nur Prozeduren, auch wenn anderes möglich wäre. Man unterscheidet zwischen *privaten* und *öffentlichen* Prozeduren. Die privaten können nur innerhalb der Bibliothek benutzt werden, öffentliche auch von Programmen, die die Bibliothek benutzen. Eine private Prozedur wird wie eine normale Prozedur mit *Procedure* geschrieben, für eine öffentliche benutzt man hingegen *ProcedureDLL*.

Nachfolgend soll nun ein einfaches Beispiel folgen.

dll-datei.pb

```
ProcedureDLL test(x.l)
  OpenConsole()
  PrintN("Ich bin eine DLL-Datei und bekam als Argument "+Str(x)+" übergeben")
  Delay(2000)
EndProcedure
```

main.pb

```
; Listing 33: DLL-Dateien

Prototype Test(x.l)
OpenLibrary(0, "./test.dll")
dllFunction.Test = GetFunction(0, "test")
dllFunction(2)
CloseLibrary(0)
```

```
Ausgabe:

Ich bin eine DLL-Datei und bekam als Argument 2 übergeben
```

Die Datei **dll-datei.pb** muss erst als .dll-Datei kompiliert werden. Dazu wählt man *Compiler->Compiler-Optionen...* und stellt hier unter *Executable-Format Shared Dll* ein und damit das *Ü* richtig dargestellt wird sollte auch *Unicode-Executable erstellen* eingehackt werden. Nun wählt man *Compiler->Executable erstellen...* und speichert die .dll-Datei am gleichen Ort wie **main.pb** ab.

Wie man sieht, wird in **dll-datei.pb** eine öffentliche Prozedur erstellt, die eine Konsole öffnet und einen String ausgibt. Nach 2 Sekunden geht die Programmausführung weiter.

In **main.pb** wird zuerst ein Prozedurprototyp deklariert. Dies ist ein Variablentyp, der eine Funktion repräsentiert. Der Prototyp muss die gleichen Argumente haben, wie die Prozedur der .dll-Datei. Nun öffnet man die Bibliothek mit *OpenLibrary()*. Das erste Argument ist hierbei eine Identifikationsnummer für die geöffnete Bibliothek. Als nächstes erstellt man eine Variable vom Typ des vorher deklarierten Prozedurprototyps mit *GetFunction()*. *dllFunction* ist hierbei ein Zeiger, was in einem späteren Kapitel erklärt wird. Soviel sei gesagt: *GetFunction()* gibt von einer bestimmten Bibliothek (hier Nummer 0), die vorher geöffnet und damit in den Arbeitsspeicher geladen wurde, die Adresse der Prozedur zurück, deren Namen *GetFunction()* übergeben bekommt. *dllFunction()* kann nun wie die Prozedur eigentliche Prozedur verwendet werden. Zuletzt schließt man die Bibliothek mit *CloseLibrary()*.

3.4 Zeiger

Viele Programmier bezeichnen Zeiger als eines der am schwierigsten zu verstehenden Themen im Bereich Programmierung. Dabei sind Zeiger ganz einfach zu verstehen, wenn man sich klar macht, wie Variablen und andere Daten im Arbeitsspeicher liegen.

Jede Variable im Arbeitsspeicher hat eine *Adresse*. Wenn man sich den Arbeitspeicher bildlich wie eine Reihe von verschieden großen Kisten (je nachdem welchen Typ die Variable hat) vorstellt, dann liegen die Variablen in diesen, wobei jeder dieser Kisten eine Adresse hat, wie eine Hausnummer. Der Computer arbeitet intern auf der Maschinensprachenebene mit diesen Adressen, um auf die Variablen zuzugeifen, er benutzt dafür nicht die Bezeichner. Man benutzt nun Zeiger um dies ebenfalls wie der Computer zu tun. Im Zeiger speichert man also die Adresse einer Variable, um über jenen auf diese zuzugreifen. Man nennt diesen Zugriff dann *Dereferenzierung*, denn der Zeiger stellt eine Referenz zur Variable dar.

```
; Listing 33: Zeiger

Procedure change(*ptr2.l)
  PokeL(*ptr2, 6)
EndProcedure

x = 5

OpenConsole()

PrintN(Str(x))

*ptr = @x
change(*ptr)
PrintN(Str(PeekL(*ptr)))

Delay(2000)
```

```
Ausgabe:

5
6
```

Man definiert eine Funktion, die einen Zeiger auf eine Longvariable (.l) als Argument hat, d.h. sie erwartet die Adresse einer Longvariable. Man definiert die Variable x und initialisiert sie mit 5. Nachdem man sie ausgegeben hat, definiert man den Zeiger **ptr* und initialisiert ihn mit der Adresse von x. Man erhält die Adresse über den Operator @. Das Sternchen gehört fest zum Bezeichner des Zeigers. In der Prozedur wird der Zeiger über *PokeL()* dereferenziert, sodass auf die eigentliche Variable x zugegriffen werden kann, mit dem Ziel in diese einen neuen Wert zu schreiben. Wenn man auf Variablen anderen Typs dereferenzieren will, muss man dementsprechen das L austauschen, genauso wie bei *Str()* und *Val()*. Man muss *PokeL()* dabei eine Adresse und einen neuen Wert übergeben. Mit *PeekL()* kann man ebenfalls einen Zeiger derefenzieren, erhält jedoch den Wert der Variable als Rückgabewert. Für das L von *PeekL()* gilt das Gleiche wie bei *PokeL()*.

3.5 GUI-Programmierung

Dieses Kapitel soll eine Einführung in die GUI-Programmierung mit PureBasic bieten. Dabei sollen nicht alle Befehle behandelt werden, wenn man jedoch die Grundlagen verstanden hat, kann man sich die restlichen Befehle je nach Bedarf über die Referenz aneignen.

```
; Listing 34: Ein Fenster

OpenWindow(0, 0, 0, 200, 200, "Ein Fenster",
#PB_Window_ScreenCentered|#PB_Window_SystemMenu)
TextGadget(0, 0, 0, 100, 20, "TextGadget")

Repeat : Until WindowEvent() = #PB_Event_CloseWindow
```

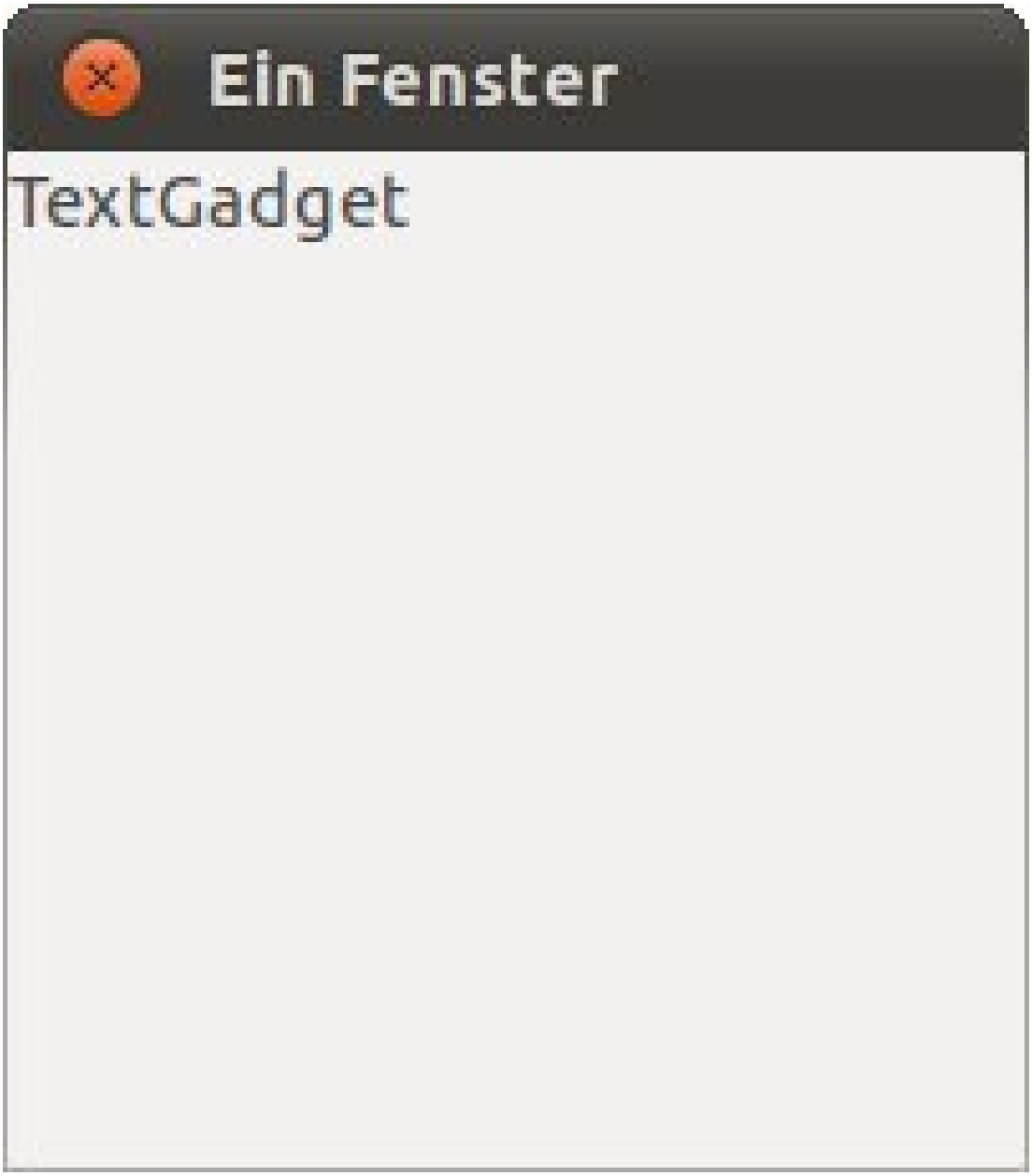

Abb. 1

Nach der Kompilierung wird ein simples Fenster mit dem Text *TextGadget* dargestellt, das nichts anderes tut außer über "x" geschlossen zu werden.

Über *OpenWindow()* wird das Fenster geöffnet, wobei die Argumente darstellen, wie das Fenster dargestellt werden soll. Von Links, nach Rechts: Fensternummer, Position des Fensters in x- und in y-Richtung, Breite und Höhe, sowie der Titel. Die x-Achse verläuft von Links nach Rechts, die y-Achse von Oben nach Unten. Die beiden Konstanten sind sogenannte *Flags*, die über ein binäres "Oder" verknüpft werden. Es ist nicht wichtig zu verstehen was das ist, wichtig ist nur, dass man so die Flags trennt. Die erste bedeutet, dass das Fenster in der Bildschirmmitte angezeigt wird und die zweite sorgt dafür, dass das Schließensymbol angezeigt wird. In der nächsten Zeile ist ein sogeanntes Textgadget. Alles in Fenstern

wird über Gadgets dargestellt. Die Argumente sind die gleichen, wie beim Fenster. Weitere Flags für Fenster und die jeweiligen Gadgets stehen in der Referenz. Die letzte Zeile ist eine Repeat-Schleife, wobei die Zeilenorientierung des Compilers über den Doppelpunkt umgangen wird. Durch ihn wird es möglich, mehrere Befehl in eine Zeile zu schreiben, was insbesondere bei solch kurzen Befehlen Sinn macht. Der Ausdruck hinter *Until* sorgt dafür, dass das Fenster erst geschlossen wird, genauer gesagt, dass das Programm erst beendet wird, denn darunter sind keine Befehle mehr, wenn das "x" geklickt wird. Dieses erzeugt nämlich ein sogenanntes Event, dass eine dynamische Programmsteuerung möglich macht. *WindowEvent()* gibt einen Wert zurück, wenn ein Event ausgelöst, der durch eine Konstante repräsentiert wird. In diesem Fall wurde also beim Klicken auf das "x" ein Event ausgelöst, das durch die Konstante *#PB_Event_CloseWindow* repräsentiert wird. Der Ausdruck wird wahr und die Schleife beendet.

3.5.1 Buttons

Buttons werden ebenfalls über Gadgets repräsentiert.

```
; Listing 35: Button

OpenWindow(0, 0, 0, 200, 200, "Ein Fenster",
#PB_Window_ScreenCentered|#PB_Window_SystemMenu)
TextGadget(0, 0, 0, 200, 20, "TextGadget")
ButtonGadget(1, 0, 30, 100, 30, "Klick mich!")

Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
      End
    Case #PB_Event_Gadget
      If EventGadget() = 1
        SetGadgetText(0, "Button geklickt")
      EndIf
  EndSelect
ForEver
```

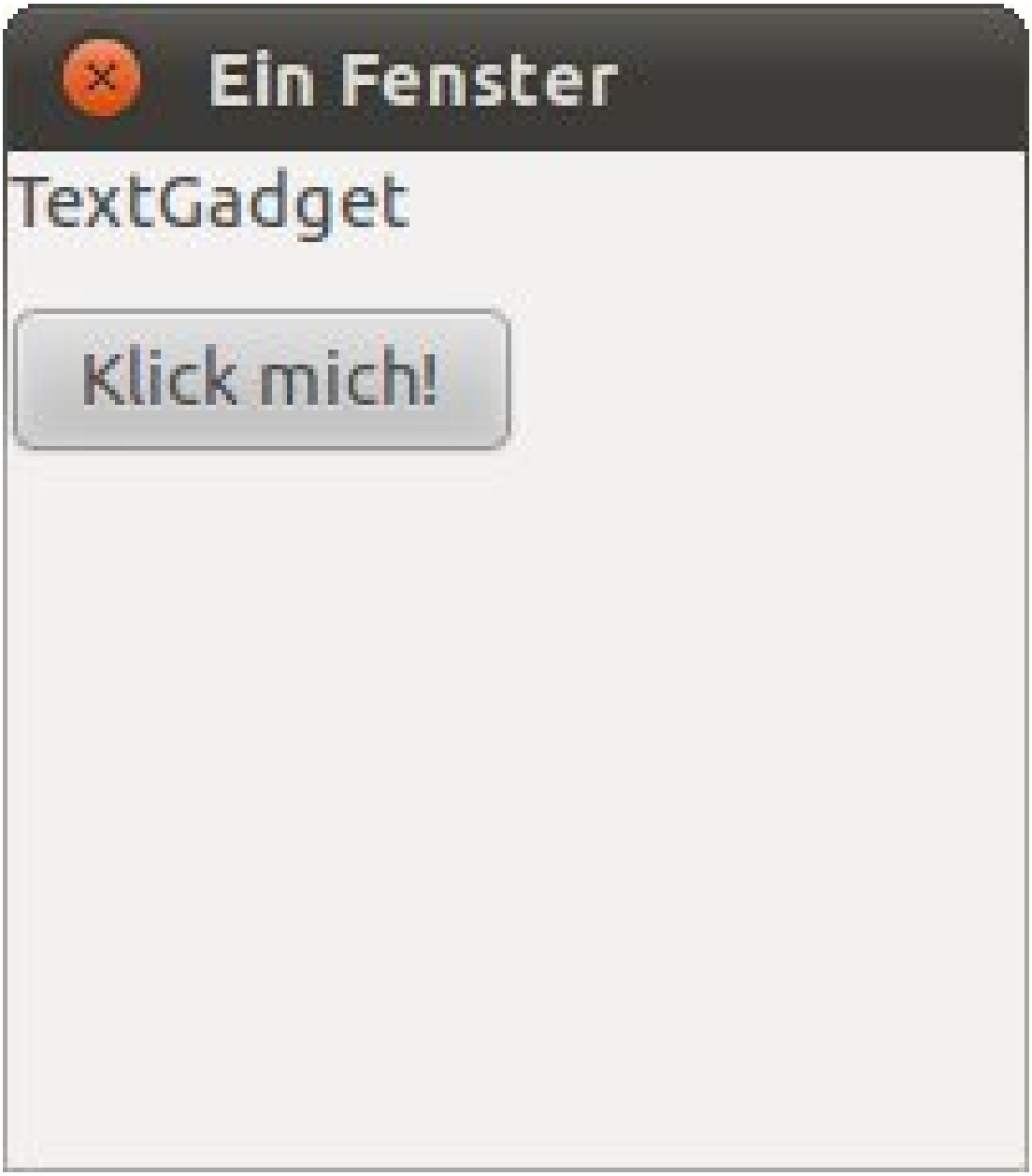

Abb. 2

Es wird genauso wie in *Listing 34* ein Fenster geöffnet, wobei noch ein Button hinzugefügt wird. Die Gadgetnummer muss dabei natürlich eine andere als die vom Textgadget sein. Wenn man nun auf den Button klickt, ändert sich der Inhalt des Textgadgets.

In der Repeat-Schleife, wird wieder auf Events reagiert. Neu ist zuerst das Schlüsselwort *ForEver*, dass dafür sorgt, dass die Repeat-Schleife nie abbricht, außer durch *Break* oder wie in diesem Fall durch *End*. Man mach sich hier die Fallunterscheidung zunutze: WindowEvent() gibt wieder einen Wert zurück, wenn ein Wert ausgelöst wird, der durch *Select* unterschieden wird. Wenn das Fenster geschlossen wurde, wird das Programm durch das Schlüsselwort *End* beendet. Wen jedoch ein Event von einem Gadget ausgelöst wurde, wird der Wert *#PB_Event_Gadget* zurückgegeben. Mit dem Befehl *EventGadget()* wird die

Gadgetnummer des auslösenden Gadgets zurückgegeben. Wenn es der Button war, wird über den Befehl *SetGadgetText()* der Text des Textgadgets geändert, wobei die Argumente die Gadgetnummer und der neue Text sind.

3.5.2 Stringgadgets

Bisher passierte noch nicht viel. Interessant wird es, sobald Stringgadgets ins Spiel kommen, mit denen man Benutzereingaben empfangen kann.

```
; Listing 36: Stringgadgets

OpenWindow(0
,200,200,200,150,"Idealgewicht",#PB_Window_SystemMenu|#PB_Window_ScreenCentered)
StringGadget(0,0,0,200,25,"Körpergröße eingeben in cm")
StringGadget(1,0,35,200,25,"Geschlecht (m/w)")
ButtonGadget(2,0,75,200,30,"Idealgewicht ausrechnen")
StringGadget(3,0,115,200,25,"Idealgewicht", #PB_String_ReadOnly)

Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
      End
    Case #PB_Event_Gadget
      If EventGadget() = 2
        If GetGadgetText(1) = "m"
          SetGadgetText(3, StrF((ValF(GetGadgetText(0))-100) * 0.93) + " bis "
+StrF((ValF(GetGadgetText(0))-100)*0.97) + " kg")
        ElseIf GetGadgetText(1) = "w"
          SetGadgetText(3, StrF((ValF(GetGadgetText(0))-100) * 0.88) + " bis "
+StrF((ValF(GetGadgetText(0))-100)*0.92) + " kg")
        Else
         MessageRequester("Fehler","Bitte echtes Geschlecht eingeben")
       EndIf
     EndIf
  EndSelect
ForEver
```

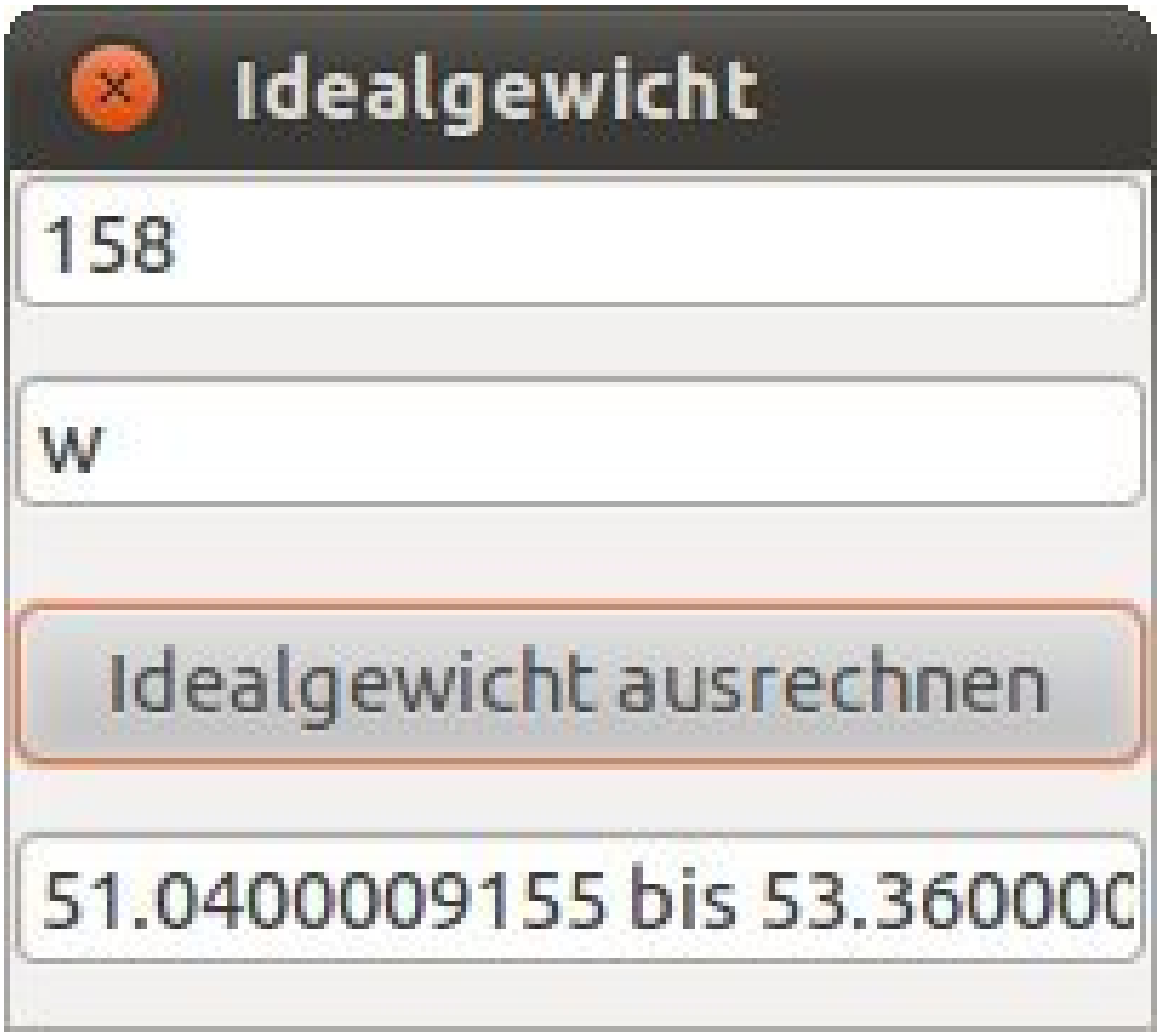

Abb. 3

Abermals wird ein Fenster geöffnet. Neu sind die Eingabefelder, die durch Stringgadgets repräsentiert werden. In diese kann man Strings eingeben, die mit *GetGadgetText()* abgerufen werden, wobei das Argument die Gadgetnummer ist. *GetGadgetText()* ist also das Gegenstück zu *SetGadgetText()*.

Das Programm tut nichts anderes, als das Idealgewicht anhand von Größe und Geschlecht auszurechnen. Sobald der Button gedrückt wurde (*EventGadget() = 2*), wird mit den genannten Befehlen abgerufen welches Geschlecht eingegeben wurde und dann mit der Körpergröße das Idealgewicht ausgerechnet, das mit *SetGadgetText()* im unteren Stringgadget angezeigt wird. Die Konstante beim unteren Stringgadget verhindert, dass in dieses Strings eingegeben werden können, es ist also lediglich zum Anzeigen von Strings da. Der letzte neue Befehl ist *MessageRequester()*. Dieser tut nichts anderes, als ein Fenster mit einem Titel und einem Text zu öffnen, das über einen Button geschlossen wird. Das erste Argument ist der Titel, das zweite der Text.

Mit diesen wenigen Beispielen hat man schon einen guten Überlick über die GUI-Programmierung gewonnen. Die wenigen anderen Elemente sind in der PureBasic-Referenz erläutert und können sich bei Bedarf angeeignet werden.

3.5.3 Enumeration und #PB_Any

Bisher wurde den Gadgets und den Fenstern eine Nummer gegeben. Es ist offensichtlich, dass wenn man mit vielen Gadgets arbeitet, dies unübersichtlich werden kann. Es liegt nahe Konstanten zu benutzen, um das Problem zu lösen. Um das Ganze noch einfacher zu gestalten gibt es das Schlüsselwort *Enumeration*

```
; Listing 37: Enumeration

Enumeration
  #Window
  #TextGadget
  #ButtonGadget
EndEnumeration

OpenWindow(#Window, 0, 0, 200, 200, "Ein Fenster",
#PB_Window_ScreenCentered|#PB_Window_SystemMenu)
TextGadget(#TextGadget, 0, 0, 100, 20, "TextGadget")
ButtonGadget(#ButtonGadget, 0, 30, 100, 30, "Klick mich!")

Repeat
 Select WindowEvent()
   Case #PB_Event_CloseWindow
     End
   Case #PB_Event_Gadget
     If EventGadget() = #ButtonGadget
       SetGadgetText(#TextGadget, "Button geklickt")
     EndIf
 EndSelect
ForEver
```

Durch *Enumeration* wird den eingeschlossenen Konstanten der Reihe nach ein Wert zugewiesen, angefangen bei 0, die nächste Konstante bekommt die 1 zugewiesen usw. Es ist außerdem möglich einen Startwert festzulegen, indem man ihn hinter *Enumeration* schreibt, und über das Schlüsselwort *Step*, wie bei der For-Schleife, die Schrittgröße festzulegen.

Es gibt noch eine weitere Möglichkeit, über die Konstante *#PB_Any*

```
; Listing 38: #PB_Any

Window = OpenWindow(#PB_Any, 0, 0, 200, 200, "Ein Fenster",
#PB_Window_ScreenCentered|#PB_Window_SystemMenu)
TextGadget = TextGadget(#PB_Any, 0, 0, 100, 20, "TextGadget")
ButtonGadget = ButtonGadget(#PB_Any, 0, 30, 100, 30, "Klick mich!")

Repeat
 Select WindowEvent()
   Case #PB_Event_CloseWindow
     End
   Case #PB_Event_Gadget
     If EventGadget() = ButtonGadget
       SetGadgetText(TextGadget, "Button geklickt")
     EndIf
```

```
 EndSelect
ForEver
```

Indem man dem Fenster und dem Gadget die Konstante #PB_Any als Kennnummer übergibt, wird ihnen automatisch eine zugewiesen. Man erwirkt damit, dass der Rückgabewert der Befehle diese Nummer ist.

3.6 Event-Handler

Dieses Kapitel geht näher auf verschiedenen Arten von Ereignissen (Event) ein und zeigt deren sinnvollen Einsatz

3.6.1 WindowEvent() / WaitWindowEvent()

Es gibt zwei Funktionen, mit welchen ein Ereignis abgefragt werden kann. Diese sind *WindowEvent()* und *WaitWindowEvent()*, wobei der einzige Unterschied darin besteht, das *WaitWindowEvent()* darauf wartet bis etwas geschieht und erst dann einen Wert ausgibt, während *WindowEvent()* bei jedem Aufruf entweder das Event, oder solange nichts passiert ist, den Wert '0' zurück gibt. WindowEvent() sollte vermieden werden, da das Programm auch wenn es nicht benutzt wird die CPU stark belastet. Die Funktion *WaitWindowEvent()* unterstütz den optionalen Parameter 'timeout' welcher zulässt das das Programm mit geringer CPU Auslastung Berechnungen durchführen kann.

```
; Listing 39: WindowEvent

OpenWindow(0,0,0,400,200,"Uhr",#PB_Window_SystemMenu|#PB_Window_ScreenCentered)
TextGadget(0,0,100,400,20,"",#PB_Text_Center)
Repeat

  Select WaitWindowEvent(20)
    Case #PB_Event_CloseWindow
      End
  EndSelect

  SetGadgetText(0,FormatDate("%hh:%ii:%ss",Date()))

ForEver
```

Der Timeout wurde hier auf 20 Millisekunden gesetzt, das heißt wenn das Programm 20ms gewartet hat gibt es wie WindowEvent() eine 0 zurück. Diese Variante hat gegenüber einer Kombination aus WindowEvent() und Delay(20) den Vorteil, das sollte vor Ablauf der 20ms etwas passieren, z.B. es wird ein Button gedrückt, kann das Programm sofort reagieren und muss nicht erst noch die 20ms abwarten bis es wieder etwas machen kann. FormatDate wird verwendet, um auf einfache Weise ein Datumswert in verständlicher Weise als String darzustellen.

3.6.2 Timer

Eine Möglichkeit regelmäßig wiederkehrende Aufgaben zu erledigen sind Timer.

```
; Listing 40: Timer

O
penWindow(0,0,0,400,100,"Timer",#PB_Window_SystemMenu|#PB_Window_ScreenCentered)

TextGadget(0,0, 0,400,20,"Dieses Programm ist nun seit")
TextGadget(1,0,20,400,20,"")
TextGadget(2,0,40,400,20,"Sekunden aktiv")

AddWindowTimer(0,0,1000)

Repeat

  Select WaitWindowEvent()
    Case #PB_Event_CloseWindow
      End
    Case #PB_Event_Timer
      sekunden+1
      SetGadgetText(1,Str(sekunden))
  EndSelect

ForEver
```

Durch *AddWindowTimer* wird im Abstand von 1000ms einmal das Event *#PB_Event_Timer* aufgerufen.

```
; Listing 41: Timer vs WaitWindowEvent

O
penWindow(0,0,0,400,100,"Timer",#PB_Window_SystemMenu|#PB_Window_ScreenCentered)

TextGadget(0,0, 0,400,40,"Fahre mit der Maus über das Fenster um den
unterschied zwischen Timer und WaitWindowEvent mit timeout zu sehen")
TextGadget(1,0,40,400,20,"")

Repeat

  Select WaitWindowEvent(1000)
    Case #PB_Event_CloseWindow
      End
  EndSelect
  counter+1
  SetGadgetText(1,Str(counter))

ForEver
```

In diesem Beispiel steigt der Counter einmal pro Sekunde, solange das Fenster inaktiv ist. Sobald etwas mit dem Fenster gemacht wird liefert *WaitWindowEvent* einen Rückgabewert und muss nicht mehr auf den Ablauf der 1000ms warten. Daher wird der Counter schon vor Ablauf der Sekunde um eins erhöht werden.

3.6.3 EventGadget

Um mehrere verschiedene Gadgets benutzen zu können benötigt man wie schon in GUI-Programmierung beschrieben *EventGadget()*. Sollte man wie es in größeren Programmen üblich ist mehrere Gadgets auf einem Fenster haben bietet es sich an diese auch über ein *Select/Case* zu verwalten.

```
; Listing 42: EventGadget

OpenWindow(0, 0, 0, 200, 200, "EventType",
#PB_Window_ScreenCentered|#PB_Window_SystemMenu)
StringGadget(0, 0, 0, 200, 20, "Schreibe irgendwas")
TextGadget(1, 0, 20, 200, 20, "Der Text hat 18 Zeichen")
TextGadget(2, 0, 50, 200, 20, "Lösche durch Klicken:")
ListViewGadget(3, 0, 70, 200, 130)
For x=0 To 20
  AddGadgetItem(3,x,"Element "+Str(x))
Next

Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
      End
    Case #PB_Event_Gadget

      Select EventGadget()
        Case 0
          SetGadgetText(1,"Der Text hat "+Str(Len(GetGadgetText(0)))+"
Zeichen")

        Case 3
          RemoveGadgetItem(3,GetGadgetState(3))

      EndSelect

  EndSelect
ForEver
```

Das *ListViewGadget* ist eine Art Container in dem eine beliebige Anzahl an Strings gespeichert werden kann. Durch die Funktion *AddGadgetItem* kann ein neues Element hinzugefügt und der *RemoveGadgetItem* wieder gelöscht werden. Wenn ein Element ausgewählt ist kann wie beim *StringGadget* der Text des Elementes über *GetGadgetText* ausgelesen werden.

3.6.4 EventType

Manchmal ist es nützlich verschiedene Aktionen des Benutzers unterscheiden zu können. So kann es durchaus vorkommen das bei einem Rechtsklick oder einem Doppelklick etwas anderes geschehen soll als bei einem normalen Links klick. Hierfür kann man *EventType* benutzen.

Hinweis: EventType ist nicht für alle Gadgets verfügbar, welche das sind entnehmen Sie bitte aus dem Referenz Handbuch.

```
; Listing 43: EventType

```

```
OpenWindow(0, 0, 0, 200, 200, "EventType",
#PB_Window_ScreenCentered|#PB_Window_SystemMenu)
StringGadget(0, 0, 0, 200, 20, "Schreibe irgendwas")
TextGadget(1, 0, 20, 200, 20, "Der Text hat 18 Zeichen")
TextGadget(2, 0, 50, 200, 20, "Lösche nur duch Doppelklick:")
ListViewGadget(3, 0, 70, 200, 130)
For x=0 To 20
  AddGadgetItem(3,x,"Element "+Str(x))
Next

Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
      End
    Case #PB_Event_Gadget

      Select EventGadget()
        Case 0
          Select EventType()
            Case #PB_EventType_Change
              SetGadgetText(1,"Der Text hat "+Str(Len(GetGadgetText(0)))+"
Zeichen")

            Case #PB_EventType_Focus
              SetGadgetText(0,"")
              SetGadgetText(1,"Der Text hat "+Str(Len(GetGadgetText(0)))+"
Zeichen")

          EndSelect

        Case 3
          Select EventType()
            Case #PB_EventType_LeftDoubleClick
              RemoveGadgetItem(3,GetGadgetState(3))
          EndSelect

      EndSelect

  EndSelect
ForEver
```

In diesem Beispiel wird gegenüber *Listing 42* unterschieden was genau der Benutzer mit dem Gadget macht. So wird im *StringGadget* nicht nur gezählt wie viele Zeichen in dem *StringGadget* enthalten sind, sondern es wird beim anklicken der vorherige Inhalt gelöscht.

3.6.5 EventMenu

```
; Listing 44: EventMenu

OpenWindow(0, 0, 0, 200, 200, "EventMenu",
#PB_Window_ScreenCentered|#PB_Window_SystemMenu)
CreateMenu(0,WindowID(0))
MenuTitle("Datei")
MenuItem(0,"Neu")
MenuItem(2,"Einstellungen")
MenuItem(3,"Beenden")
MenuTitle("?")
MenuItem(4,"Hilfe")

CreateToolBar(0,WindowID(0))
ToolBarStandardButton(0,#PB_ToolBarIcon_New)
```

```
ToolBarStandardButton(1,#PB_ToolBarIcon_Help)
ToolBarStandardButton(3,#PB_ToolBarIcon_Delete)

Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
      End
    Case #PB_Event_Menu

      Select EventMenu()
        Case 0
          MessageRequester("Neu","Es wurde 'Neu' angeklickt!")
        Case 1
          MessageRequester("Hilfe","Es wurde 'Hilfe' angeklickt!")
        Case 2
          MessageRequester("Einstellungen","Es wurde 'Einstellungen'
angeklickt!")
        Case 3
          MessageRequester("Beenden","Es wurde 'Beenden' angeklickt!")
        Case 4
          MessageRequester("Hilfe","Es wurde 'Hilfe' angeklickt!")
      EndSelect

  EndSelect
ForEver
```

Durch *CreateMenu* wird eine neue Menüleiste im durch *WindowID* bestimmten Fenster erstellt. Diese Werden durch den Befehl *MenuTitle* in einzelne Gruppen untergliedert, welche einzelne Elemente (*MenuItem*) enthalten. Ähnlich wie bei den Gadgets werden hier Konstanten verwendet mit denen die einzelnen Elemente später über *EventMenu* identifiziert werden können. Jedoch unterscheiden sich *MenuItems* von Gadgets in zwei Punkten. Erstens darf hier nicht der Wert *#PB_Any* verwendet werden, und zweitens können die Konstanten hier mehrfach vorkommen. Es muss aber beachtet werden, dass die MenuID für alle Elemente die selbe Funktion hat, da nicht unterschieden wird ob ein Menü über die Menüleiste oder über die Toolbar aktiviert worden ist.

3.7 SizeWindow

SizeWindow ist nützlich wenn das Fenster in der Größe verändert wurde, z.B. Maximiert, und der Inhalt automatisch angepasst werden soll.

```
; Listing 45: SizeWindow

OpenWindow(0, 0, 0, 200, 200, "SizeWindow", #PB_Window_Scree
nCentered|#PB_Window_SystemMenu|#PB_Window_MaximizeGadget|#PB_Window_SizeGadget)
WindowBounds(0, 200, 200, #PB_Ignore, #PB_Ignore)
ListViewGadget(0, 0, 0, 200, 180)
ButtonGadget(1, 0, 180, 200, 20, "Beenden")
For x=0 To 50
  AddGadgetItem(0,x,"Element "+Str(x))
Next

Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
      End
```

```
    Case #PB_Event_SizeWindow
      ResizeGadget(0,0,0,WindowWidth(0),WindowHeight(0)-20)
      ResizeGadget(1,WindowWidth(0)/2-100,WindowHeight(0)-20,200,20)

    Case #PB_Event_Gadget

      Select EventGadget()
        Case 0
          End

      EndSelect

  EndSelect
ForEver
```

Die Parameter *#PB_ Window_ MaximizeGadget* und *#PB_ Window_ SizeGadget* werden benötigt um dem Benutzer das Ändern der Fenstergröße zu erlauben. Mit *WindowBounds* wird sichergestellt, das der Benutzer das Fenster nicht kleiner als 200x200 Pixel verkleinern kann, da der Inhalt dann nicht mehr übersichtlich dargestellt werden kann. Durch *ResizeGadget* kann die Größe und Position des Gadgets verändert werden. Die Position zu verändern kann in solchen Fällen nützlich sein, wenn ein Button an der Unterseite des Fensters fixiert, aber nicht die Größe verändert werden soll.

3.8 Zeichnen

Es ist möglich Formen und Linien zu zeichnen. Dies ist z.B. nützlich, wenn man ein Simulationsprogramm programmerien will oder einen Funktionsplotter. Plakativ soll hier zweidimensional gezeichnet werden, in der PureBasic-Vollversion ist es auch möglich 3D-Objekte zu erstellen.

```
; Listing 46: Zeichnen

Procedure.f turn_x(x.f, y.f, degree.f)
  ProcedureReturn Cos(Radian(degree))*x-Sin(Radian(degree))*y
EndProcedure

Procedure.f turn_y(x.f,y.f,degree.f)
  ProcedureReturn Sin(Radian(degree))*x+Cos(Radian(degree))*y
EndProcedure

x1.f=100
y1.f=100
x2.f=140
y2.f=140
x3.f=100
y3.f=140
degree.f = 1

OpenWin
dow(0,0,0,400,400,"Umkehrungen",#PB_Window_SystemMenu|#PB_Window_ScreenCentered)

AddWindowTimer(0,0,10)
Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
```

```
      End

    Case #PB_Event_Timer

      StartDrawing(WindowOutput(0))

      Box(0,0,400,400)
      LineXY(x1+200,y1+200,x2+200,y2+200,RGB(0,0,0))
      LineXY(x1+200,y1+200,x3+200,y3+200,RGB(0,0,0))
      LineXY(x2+200,y2+200,x3+200,y3+200,RGB(0,0,0))
      LineXY(x1+200,y1+200,200,200,RGB(0,0,0))
      Circle(200,200,2,RGB(0,0,0))

      x1_new.f=turn_x(x1,y1,degree)
      y1_new.f=turn_y(x1,y1,degree)
      x2_new.f=turn_x(x2,y2,degree)
      y2_new.f=turn_y(x2,y2,degree)
      x3_new.f=turn_x(x3,y3,degree)
      y3_new.f=turn_y(x3,y3,degree)

      x1=x1_new
      y1=y1_new
      x2=x2_new
      y2=y2_new
      x3=x3_new
      y3=y3_new

      StopDrawing()

  EndSelect

ForEver
```

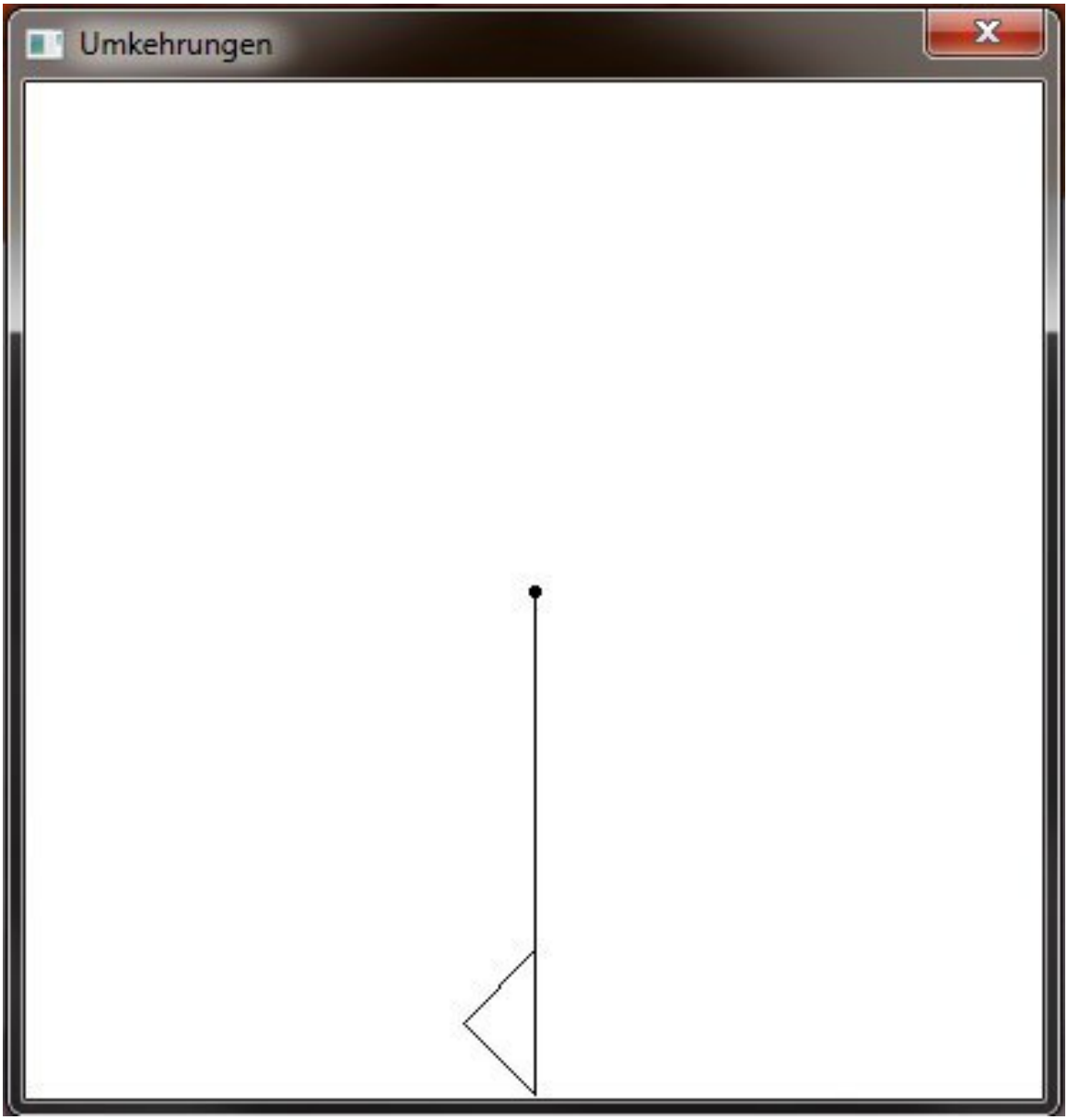

Abb. 4

Dieses Programm öffnet ein Fenster, in welchem in der Mitte ein Kreis gezeichnet wird (*Circle()*), von dem eine Linie ausgeht, an der ein Dreieck hängt (*LineXY()*). Die Linie und das Dreieck drehen sich nun im Uhrzeigersinn.

Mit *StartDrawing()* wird ausgesagt, dass nun begonnen werden soll etwas zu zeichnen. Das Argument ist dabei die Form der Ausgabe, in diesem Fall ein Fenster. *WindowOutput()* muss wiederum als Argument die Fensternummer übergeben werden, die zur Zeichenausgabe dienen soll. Für *StartDrawing()* gibt es noch weitere Argumente, um z.B. eine Ausgabe zu drucken. Wenn fertig gezeichnet wurde und man noch andere Dinge im Programm ausführen will, sollte man *StopDrawing()* ausführen, da damit nicht mehr benötigte Rechnerressourcen freigegeben werden. Mit *Circle()* wird wie gesagt ein Kreis gezeichnet, wobei die ersten beiden Argumente die x- bzw. die y-Position darstellen und die anderen Argumente den

Radius und die Farbe in RGB-Schreibweise. Letzteres heißt, dass die erste Zahl den Rot-, die zweite den Grün- und die letzte den Blauanteil der Farbe darstellt. In der Referenz ist eine Farbtabelle enthalten, über die man die Werte für bestimmte Farben herausfinden kann. Bei der x-y-Position wird 200 hinzuaddiert, weil die Prozeduren zum Drehen des Zeigers, diesen um die linke obere Fensterecke drehen. Mit *LineXY()* werden die Linien gezeichnet, wobei man jeweils zwei Punkte, die verbunden werden sollen, mit x-y-Koordinate angeben muss. Zusätzlich muss wieder die Farbe für die Linien angegeben werden. Darunter werden die Punkte, die verbunden werden sollen, gedreht. Die Prozeduren basieren hierbei auf Vektorrechnung. Die neuen Werte müssen zuerst in Zwischenvariablen gespeichert werden, da sowohl für die neue x-, als auch für die neue y-Koordinate, die alten Koordinaten benötigt werden. Mit *Box()* wird der gesamte bisher gezeichnete Inhalt übermalt, in dem über alles ein großes Viereck gelegt wird, da sonst die alten Linien immer noch sichtbar wären.

3.8.1 Farben

Auch das Benutzen von Farben ist ohne weiteres möglich.

```
; Listing 47: Farben

Op
enWindow(0,0,0,400,400,"Farben",#PB_Window_SystemMenu|#PB_Window_ScreenCentered)

rot   = RGB(255,  0,  0)
gruen = RGB(  0,255,  0)
blau  = RGB(  0,  0,255)
gelb  = RGB(255,255,  0)

Repeat
  Select WindowEvent()
    Case #PB_Event_CloseWindow
      End
    Case #PB_Event_Repaint

      StartDrawing(WindowOutput(0))

      Box(  0,  0,200,200,rot)
      Box(200,  0,200,200,gruen)
      Box(  0,200,200,200,blau)
      Box(200,200,200,200,gelb)

      StopDrawing()

  EndSelect

ForEver
```

Dieses Programm öffnet ein Fenster, in dem vier Quadrate in verschiedenen Farben gezeichnet werden.

Zunächst werden die Werte der vier Farben rot,grün,blau und gelb über die Funktion RGB() in den Variablen gespeichert. Diese werden Benutzt um den Boxen einen Farbwert zuzuweisen. Alternativ können die RGB-Werte, wie im vorhergehenden Beispiel, auch direkt als letzer Parameter der Zeichenfunktion (hier: Box()) geschrieben werden.

Die Farben können alternativ auch als Hexadezimalzahl oder als Dezimalzahl geschrieben werden.

```
blau = RGB(0,0,255)
blau = $FF0000
blau = 16711680
```

Jede dieser drei Zeilen bewirkt das selbe. Die Dezimalschreibweise ist nicht geläufig, da aus ihr nicht auf Anhieb abzulesen ist welche Farbe hinter der Zahl steckt. Einfacher ist es bei *rot*, die den Dezimalwert 255 hat. Die Funktion *RGB()* geht grob nach folgender Rechnung vor:

```
Rückgabewert = Rot + 2^8 * Grün + 2^16 * Blau
```

4 Objektorientierung

4.1 Installation von PureObject

Das Plugin PureObject ist für Windows und Linux verfügbar und stellt objektorientierte Programmierung als sogenannter Präprozessor zur Verfügung. Präprozessor heißt, das der objektorientierte Code, der standardmäßig nicht verfügbar ist, in normalen PureBasic-Code umgewandelt wird, damit der Compiler letztendlich den Code übersetzen kann.

Dieses Kapitel basiert auf der offiziellen Anleitung[1].

PureObject lädt man sich hier[2] herunter. Nachdem man das Paket entpackt hat, klickt man in der IDE auf *Werkzeuge->Werkzeuge konfigurieren....* Hier legt man nun mit *Neu* einen neuen Eintrag an. Nun klickt man bei *Ereignis zum Auslösen des Werkzeugs* die Option *Vor dem kompilieren/Starten* an. Im rechten Kasten klickt man *Warten bis zum Beenden des Werkzeugs*, *Versteckt starten* und *Werkzeug vom Hauptmenü verstecken.* Im Eingabefeld *Kommandozeile* muss der Pfad zur *oop*-Datei angegeben werden, die man am Besten in das PureBasic-Verzeichnis kopiert. In das Feld *Argumente* schreibt man *"%FILE" "%COMPILEFILE"*. Die Anführungsstriche sind hierbei mitzuschreiben. Bei Name kann man dem Werkzeug einen sinnvollen Namen geben, z.B. "OOP-Präprozessor 1".

Als nächstes legt man einen zweiten Eintrag an. Es werden genau die gleichen Dinge angegeben, mit der Ausnahme, dass bei *Ereignis zum Auslösen des Werkzeugs* die Option *Vor dem Erstellen des Executable* gewählt wird.

Unter *Datei->Einstellungen* sucht man den Eintrag *Eigene Schlüsselwörter*. Dort gibt man folgende Schlüsselwörter ein:

- Abstract
- Class
- DeleteObject
- EndClass
- Fixed
- Flex
- Force
- NewObject
- This

Soviel zur Installation.

---

1 http://pb-oop.origo.ethz.ch/wiki/IDE_Installation ^{http://pb-oop.origo.ethz.ch/wiki/IDE_Installation}

2 http://pb-oop.origo.ethz.ch/wiki/pureobject

4.1.1 Quellen

4.2 Einstieg in die Objektorientierte Programmierung

Die objektorientierte Programmierung basiert auf dem Paradigma der Objektorientierung. Ein Paradigma gibt an, in welchem Programmierstil der Programmierer an eine Problemstellung heran geht. PureBasic z.B. ist vorallem *imperativ* und *strukturiert*, d.h. es werden Befehle in einer festen Reihenfolge verarbeitet und die Problemstellung wird in mehrere kleine Teilprobleme zerlegt (Stichwort *Procedure*), die für sich bearbeitet werden. Daraus ergibt sich die Lösung des gesamten Problems.

Bei der Objektorientierung versucht man das Problem in sogenannten Objekten darzustellen. Ein Objekt hat Eigenschaften (*Attribute*) und *Methoden*, um mit der Außenwelt, also anderen Objekten, in Kontakt zu treten. Die Attribute sind dabei meistens von der Außenwelt abgeschnitten und die Methoden stellen Schnittstellen bereit, um auf diese zuzugreifen. Ähnliche Objekte werden in sogenannten *Klassen* zusammengefasst, in denen Attribute und Methoden deklariert und definiert werden. Man könnte also z.B. alle Menschen in einer Klasse zusammenfassen.

```
; Listing 40: Klassen

OpenConsole()

Class Mensch
  Mensch(alter.l = 0, name.s = "")
  Release()

  set_alter(alter.l)
  get_alter.l()
  set_name(name.s)
  get_name.s()

  alter.l
  name.s
EndClass

Procedure Mensch\Mensch(alter.l = 0, name.s = "")
  PrintN("Ich bin der Konstruktor")
  This\alter = alter
  This\name = name
EndProcedure

Procedure Mensch\Release()
  PrintN("Ich bin der Destruktor")
EndProcedure

Procedure Mensch\set_alter(alter.l)
  This\alter = alter
EndProcedure

Procedure.l Mensch\get_alter()
   ProcedureReturn This\alter
EndProcedure

Procedure Mensch\set_name(name.s)
  This\name = name
EndProcedure
```

```
Procedure.s Mensch\get_name()
  ProcedureReturn This\name
EndProcedure

*jesus.Mensch = NewObject Mensch(2011, "Jesus")

PrintN("Name: "+*jesus\get_name())
PrintN("Alter: "+Str(*jesus\get_alter()))

DeleteObject *jesus

Delay(2000)
```

```
Ausgabe:

Ich bin der Konstruktor
Name: Jesus
Alter: 2011
Ich bin der Destruktor
```

Ganz am Anfang wird die Klasse durch das Schlüsselwort *Class* deklariert. Die Konvention lauten, dass Bezeichner groß geschrieben werden sollten. Danach folgt die Deklaration der Konstruktor- und Destruktor-Methode. Diese sind dazu da, um ein Objekt letztendlich zu initialisieren bzw. es wieder zu löschen. Dabei werden der Konstruktor-Methode zwei Argumente übergeben. Das = steht für einen Standardwert, falls kein Argument beim Methodenaufruf übergeben wird. Die Konstruktor-Methode hat dabei immer den Namen der Klasse, die Destruktor-Methode heißt immer *Release()*. Darauf folgen die Methoden, in diesem Fall Methoden, um auf die Attribute zuzugreifen. Attribute sind in PureBasic immer *privat*, d.h. für die Außenwelt nicht sichbar, in Gegensatz zu den Methoden, die *öffentlich* sind. Deshalb benötigt man diese *Setter* bzw. *Getter*. Zuletzt sei gesagt. dass Methoden immer außerhalb der Klasse definiert werden. Als letztes deklariert man die Attribute.

Bei der Methodendefinition schreibt man immer *Klassenname\Methodenname*. Der Backslash kann auch ein Punkt sein, ersterer fügt sich jedoch besser in die allgemeine PureBasic-Syntax ein. Weiter unten sieht man dann, wie auf Attribute zugegriffen wird: Die Attribute sind für die Methoden der gleichen Klasse sichtbar. Man schreibt *This\Attributname*. Auch hier gibt es wieder eine alternative Schreibweise: Anstatt des Backslash ein Pfeil (->) oder einfach nur der Pfeil, also auch ohne das *This*.

Ein Objekt wird mit dem Schlüsselwort *NewObject* gefolgt vom Konstruktor erstellt, dem die Argumente für die Konstruktormethode übergegen werden können. Bei diesem Objekt spricht man auch von einer Instanz der Klasse. Dabei wird eine Adresse auf das Objekt zurückgegeben, die in einem Zeiger vom Typ der Klasse gespeichert wird. Die Methoden werden dann über den Zeiger aufgerufen, indem man schreibt **ptr\methode()*. Zuletzt löscht man das Objekt wieder, wobei die Destruktor-Methode automatisch aufgerufen wird. Die Destruktor-Methode muss übrigens nicht definiert werden, wenn nichts beim Löschen des Objekts geschehen soll, in diesem Fall geschah dies nur zur Verdeutlichung.

4.2.1 Aufgaben

1. Es soll ein Programm geschrieben werden, dass ein Bankkonto simuliert. Es soll einen Kontostand geben und die Möglichkeit Geld abzuheben und zu hinterlegen.

4.3 Vererbung

Was tut man, wenn man einen Menschen hat, der Kung-Fu kämpfen kann, also etwas tun kann, dass nicht jeder Mensch kann. Jetzt könnte man natürlich eine komplett neue Klasse anlegen, dies ist aber ineffizient. Hier kommt *Vererbung* ins Spiel. Der Kämpfer hat natürlich noch auch alle Eigenschaften eines normalen Menschens, also ein Alter und einen Namen. Man erstellt nun eine Klasse die von der ersten erbt.

```
; Listing 41: Vererbung

Class Mensch
  Mensch(alter.l = 0, name.s = "")
  Release()

  set_alter(alter.l)
  get_alter.l()
  set_name(name.s)
  get_name.s()

  alter.l
  name.s
EndClass

Procedure Mensch\Mensch(alter.l = 0, name.s = "")
  This\alter = alter
  This\name = name
EndProcedure

Procedure Mensch\set_alter(alter.l)
  This\alter = alter
EndProcedure

Procedure.l Mensch\get_alter()
  ProcedureReturn This\alter
EndProcedure

Procedure Mensch\set_name(name.s)
  This\name = name
EndProcedure

Procedure.s Mensch\get_name()
  ProcedureReturn This\name
EndProcedure

Class Kaempfer Extends Mensch
  Kaempfer()
  Release()

  kaempfen()
EndClass

Procedure Kaempfer\Kaempfer()
  This\Base(70, "Bruce")
EndProcedure

Procedure Kaempfer\kaempfen()
  PrintN("Hai Ya!")
EndProcedure

*bruce.Kaempfer = NewObject Kaempfer()

OpenConsole()
```

```
PrintN("Name: "+*bruce\get_name())
PrintN("Alter: "+Str(*bruce\get_alter()))
*bruce\kaempfen()

DeleteObject *bruce

Delay(2000)
```

```
Ausgabe:

Name: Bruce
Alter: 70
Hai Ya!
```

Die Vererbung wird durch das Schlüsselwort *Extend* eingeleitet, hinter das man die Elternklasse schreibt. Die Klassendeklaration funktioniert dann wieder auf die gleiche Weise. Bei der Methodendefinition ist zu beachten, dass der Konstruktor den Konstruktor der Elternklasse aufrufen muss, da diesem Argumente übergeben werden sollen. Wenn der Konstruktor der Elternklasse Argumente ohne Standartwert erwartet, ist dieser Aufruf obligatorisch! Man sieht, dass die Objektinitiliasierung ebenfalls gleich ist. Außerdem kann über den Zeiger auch auf die Methoden der Elternklasse zugegriffen werden. Vererbung ist also ein sinnvoller Weg, um viele Code-Zeilen zu sparen.

4.4 Polymorphismen

Es gibt Fälle, in denen soll eine erbende Klasse eine Methode neu definieren. Wenn man bei den vorhergegangen Beispiel der Menschen bleibt, könnte es passieren das der Ausgangsmensch Deutsch spricht, der neue jedoch Englisch. Man könnte nun auf die Idee kommen, die alte Methode einfach zu überschreiben.

```
; Listing 42: Polymorphismen

OpenConsole()

Class Mensch
  Mensch()
  Release()

  sprechen()
EndClass

Procedure Mensch\Mensch()
EndProcedure

Procedure Mensch\sprechen()
  PrintN("Hallo, ich bin ein Mensch")
EndProcedure

Class English Extends Mensch
  English()
  Release()

  sprechen()
EndClass
```

```
Procedure English\English()
EndProcedure

Procedure English\sprechen()
  PrintN("Hello, I'm a human being")
EndProcedure

*obj.English = NewObject English()

*obj\sprechen()

DeleteObject *obj

Delay(2000)
```

```
Ausgabe:

Hallo, ich bin ein Mensch
```

Wie man sieht, wird die Methode der Elternklasse aufgerufen, obwohl das Objekt von Typ *English* ist. Dieses Verhalten ist so gewollt. Möchte man jedoch, dass die Methode der erbenden Klasse aufgerufen wird, kommen nun die *Polymorphismen* ins Spiel. Polymorphie bedeutet, dass ein Ausdruck mit gleichem Bezeichner *kontextabhängig* interpretiert wird. Hier wurde durch den Bezeichner *sprechen()* die Methode der Elternklasse aufgerufen. PureObject stellt nun verschiedene Schlüsselwörter bereit, um dieses Verhalten zu ändern.

4.4.1 Flex und Force

Als erstes gibt es die Möglichkeit die Methode der Elternklasse *flexibel* zu gestalten, d.h. sie kann bei Bedarf von einer erbenden Klasse überschrieben werden.

```
; Listing 43: Flex

OpenConsole()

Class Mensch
  Mensch()
  Release()

  Flex sprechen()
EndClass

Procedure Mensch\Mensch()
EndProcedure

Procedure Mensch\sprechen()
  PrintN("Hallo, ich bin ein Mensch")
EndProcedure

Class English Extends Mensch
  English()
  Release()

  sprechen()
EndClass

Procedure English\English()
```

```
EndProcedure

Procedure English\sprechen()
  PrintN("Hello, I'm a human being")
EndProcedure

*obj.English = NewObject English()

*obj\sprechen()

DeleteObject *obj

Delay(2000)
```

```
Ausgabe:

Hello, I'm a human being
```

Durch das Schlüsselwort *Flex* vor der Methode *sprechen()* der Elternklasse wird dem Compiler gesagt, dass die Methode überschrieben werden kann.

```
; Listing 44: Force

OpenConsole()

Class Mensch
  Mensch()
  Release()

  sprechen()
EndClass

Procedure Mensch\Mensch()
EndProcedure

Procedure Mensch\sprechen()
  PrintN("Hallo, ich bin ein Mensch")
EndProcedure

Class English Extends Mensch
  English()
  Release()

  Force sprechen()
EndClass

Procedure English\English()
EndProcedure

Procedure English\sprechen()
  PrintN("Hello, I'm a human being")
EndProcedure

*obj.English = NewObject English()

*obj\sprechen()

DeleteObject *obj

Delay(2000)
```

```
Ausgabe:

Hello, I'm a human being
```

Die zweite Methode sieht man hier: Durch das Schlüsselwort *Force* zwingt man den Compiler, die Elternmethode zu überschreiben. Würde man vor letztere das Schlüsselwort *Fixed* schreiben, würde *Force* einen Compilerfehler auslösen.

4.4.2 Abstrakte Methoden

Als letztes ist es möglich eine sogenannte *Abstrakte Klasse* zu schreiben. Bei dieser ist es obligatorisch, dass die Methoden überschrieben werden. Man kann sich die abstrakte Klasse wie ein Schema vorstellen, an dem man sich orientieren muss.

```
; Listing 45: Flex

OpenConsole()

Class Abstract Mensch
  sprechen()
EndClass

Class English Extends Mensch
  English()
  Release()

  sprechen()
EndClass

Procedure English\English()
EndProcedure

Procedure English\sprechen()
  PrintN("Hello, I'm a human being")
EndProcedure

*obj.English = NewObject English()

*obj\sprechen()

DeleteObject *obj

Delay(2000)
```

```
Ausgabe:

Hello, I'm a human being
```

Durch das Schlüsselwort *Abstract* wird ausgesagt, dass die nachfolgende Klassendeklaration abstrakt ist, d.h. jeder Mensch muss eine Methode *sprechen()* implemetieren. Die Klasse *English* erbt von dieser absrakten Klasse und muss folglich die Methode *sprechen()* erhalten und definieren. Man beachte, dass die abstrakte Klasse keinen Konstruktor oder Destruktor hat, folglich kann man keine Objekte aus diesen Klassen erstellen. Auch werden die Methoden dieser Klasse nicht definiert.

5 Aufgabenlösungen

5.1 Grundlagen-Lösungen

5.1.1 Hello, world!

1. Hierfür gibt es keine Lösung, das liegt allein am Ehrgeiz des Programmiers.
2. Dies sind die Fehler im Listing:
   a) Es darf immer nur ein Befehl pro Zeile stehen.
   b) *PrintN()* erwartet einen String, es fehlen also die Anführungsstriche
   c) Hinter einem Befehl stehen immer geschlossene Klammern, *Delay()* fehlt also eine

5.1.2 Variablentypen

1. Dies ist eine mögliche Lösung:

```
OpenConsole()
Print("Geben Sie den Radius des Kreises ein: ")
radius.f = ValF(Input())

PrintN("Der Umfang beträgt: "+StrF(2*radius*#Pi))

Delay(2000)
```

Diese Aufgabe ist einfach: Es gibt eine Variable *radius* vom Typ Float, was durch das kleine f hinter dem Punkt dargestellt wird. Mit *ValF(Input())* kann man einen Radius eingeben, der direkt in einen Float umgewandelt wird. Der Umfang wird direkt in der Ausgabe berechnet, wobei Pi über eine PureBasic-interne Konstante dargestellt wird, die im Kapitel vorgestellt wurde.

1.
2. Dies ist eine mögliche Lösung:

```
OpenConsole()

Print("Geben Sie einen Winkel ein: ")
winkel.f = ValF(Input())
PrintN("Der Sinus von "+StrF(winkel)+" lauten: "+StrF(Sin(Radian(winkel))))

Delay(2000)
```

5.1.3 Bedingungen

1. Dies ist eine mögliche Lösung:

```
OpenConsole()
Print("Geben Sie den Radius des Kreises ein: ")
radius.f = ValF(Input())

If radius < 0
  PrintN("Der Umfang beträgt: 0")
Else
  PrintN("Der Umfang beträgt: "+StrF(2*radius*#Pi))
EndIf

Delay(2000)
```

Die Aufgabe aus dem vorigen Kapitel wurde durch eine If-Else-Klausel erweitert, die überprüft, ob die Eingabe kleiner als 0 ist. Ansonsten arbeitet das Programm wie das erste.

1.
2. Dies ist eine mögliche Lösung:

```
OpenConsole()
Print("Geben Sie die Geschwindigkeit des Autos ein: ")
geschwindigkeit.f = ValF(Input())
#Limit = 50
If geschwindigkeit <= 50
  PrintN("Keine Geschwindigkeitsübertretung.")
Else
  PrintN("Es werden "+StrF(#Limit-geschwindigkeit)+" zu schnell gefahren.")
EndIf

Delay(2000)
```

Die Aufgabe gleicht stark der ersten: Es wird die Geschwindigkeit abgefragt, danach wird noch eine Konstante angelegt, die das Geschwindigkeitslimit darstellt. Wenn die Geschwindigkeit nicht größer als 50 km/h ist, liegt keine Übertretung vor, ansonsten schon. Die Substraktion rechnet aus, wieviel zu schnell gefahren wurde.

5.1.4 Schleifen

1. Dies ist eine mögliche Lösung:

```
OpenConsole()
Print("Geben Sie einen String ein: ")
string.s = Input()
Print("Geben Sie eine ganze positive Zahl ein: ")
zahl = Val(Input())

If zahl < 1
  PrintN("Die Zahl muss größer als 0 sein!")
Else
  For k = 1 To zahl
    PrintN(string)
  Next
EndIf

Delay(2000)
```

Die Aufgabe kann einfach über eine For-Schleife gelöst werden, wobei die Anzahl der Iterationen vorher durch die Eingabe abgefragt wurde, ebenso wie der String, der ausgegeben werden soll. Die Anzahl der Iterationen muss dabei mindestens 1 sein.

1.
2. Dies ist eine mögliche Lösung:

```
OpenConsole()

While 1
  string.s = Input()
  If string = ""
    Break
  Else
    PrintN(string)
  EndIf
Wend

Delay(2000)
```

5.1.5 Arrays, Listen und Maps

1. Dies ist eine mögliche Lösung:

```
OpenConsole()

Dim feld(4,4)

Repeat
  PrintN("1. Feld anzeigen")
  PrintN("2. Koordinate eingeben")
  PrintN("3. Programm beenden")
  Print("Auswahl: ")
  auswahl = Val(Input())

  Select  auswahl
    Case 1
      For k = 0 To 4
        For l = 0 To 4
          Print(Str(feld(k,l)))
        Next
        PrintN("")
      Next
    Case 2
      Print("x-Koordinate: ")
      x = Val(Input())
      Print("y-Koordinate: ")
      y = Val(Input())
      feld(x-1,y-1) + 1
  EndSelect

  PrintN("")
Until auswahl = 3

Delay(2000)
```

Es wird ein zweidimensionales Array angelegt, das das Feld repräsentiert. In der Repeat-Schleife wird das Menü dargestellt und danach eine Eingabe erwartet. Auf diese wird dementsprechend reagiert: Bei 1 wird das Feld über eine verschachtelte For-Schleife ausgegeben,

bei 2 müssen Koordinaten eingeben werden, die genutzt werden, um auf das Array zuzugreifen, und bei 3 bricht die Schleife ab.

5.2 Fortgeschrittene Themen-Lösungen

5.2.1 Prozeduren

1. Eine mögliche Lösung ist:

```
OpenConsole()
Procedure ausgabe(Array zahlen.l(), mittel.l, laenge.l)
  For k = 0 To laenge - 1
    If zahlen(k) > mittel
      PrintN(Str(zahlen(k)))
    EndIf
  Next
EndProcedure

Dim a(10)
For k = 0 To 9
  a(k) = k
  summe+k
Next
ausgabe(a(), summe/10, 10)

Delay(2000)
```

Es wird ein Array mit 10 Elementen angelegt und in der For-Schleife mit den Zahlen von 1 bis 10 gefüllt. Gleichzeitig werden diese Zahlen aufaddiert. Dann wird die Prozedur aufgerufen, mit dem Array, dem Mittelwert und der Anzahl der Elemente des Arrays als Argumente. In der Prozedur wird dann das Array durchlaufen und überprüft, ob das aktuelle Element größer ist als der Mittelwert.

1.
2. Eine mögliche rekursive Lösung lauten:

```
OpenConsole()

Procedure.l fib(n)
  If n <= 0
    ProcedureReturn 0
  ElseIf n = 1
    ProcedureReturn 1
  Else
    ProcedureReturn fib(n-1) + fib(n-2)
  EndIf
EndProcedure

Print("Welche Stelle soll berechnet werden: ")
n = Val(Input())
PrintN("Das Ergebnis ist "+Str(fib(n)))

Delay(2000)
```

Wie in der Aufgabe erwähnt lautet die nullte Stelle 0 und die erste 1. Dies stellt die Abbruchbedingung dar. Alle anderen Elemente berechnen sich aus der Summe der beiden vorigen. Dies wird durch die Rekursivität der Prozedur ausgedrückt.

5.3 Objektorientierung-Lösungen

5.3.1 Einstieg in die Objektorientierte Programmierung

1. Dies ist eine mögliche Lösung:

```
OpenConsole()

Class Konto
  Konto(anfang.l)
  Release()

  abheben(x.l)
  hinterlegen(x.l)
  kontostand.l()

  kontostand.l
EndClass

Procedure Konto\Konto(anfang.l)
  This\kontostand = anfang
EndProcedure

Procedure Konto\abheben(x.l)
  This\kontostand - x
EndProcedure

Procedure Konto\hinterlegen(x.l)
  This\kontostand + x
EndProcedure

Procedure.l Konto\kontostand()
  ProcedureReturn kontostand
EndProcedure

*konto.Konto = NewObject Konto(100)

*konto\abheben(50)
*konto\hinterlegen(20)
PrintN("Aktueller Kontostand: "+Str(*konto\kontostand))

DeleteObject *konto

Delay(2000)
```

Es wird eine Klasse angelegt, die ein Konto darstellt. Sie hat einen Konstruktor, dem man den Anfangskontostand übergeben muss. Außerdem kann man Geld abheben (x wird vom Kontostand abgezogen) und hinterlegen (x wird zum Kontostand hinzuaddiert). Außerdem kann man den Kontostand mit *Konto\kontostand()* abfragen. Um zu zeigen, wie sich die Methoden auswirken, wird ein Konto angelegt, indem man eine Instanz der Klasse *Konto* erstellt und hebt etwas Geld ab und hinterlegt etwas. Als Ausgabe erhält man 70.

6 Beispielprogramme

6.1 sFTPc

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  sFTPc - a simple FTP client
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
InitNetwork()

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Structure to handle FTP-connections
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Structure connection
  id.l
  adress.s
  user.s
  pass.s
  port.l
EndStructure

Declare establishConnection(List connections.connection())
Declare checkConnection(n.l)
Declare closeConnection(List connections.connection())
Declare fileManager(n.l)
Declare showDir(n.l)
Declare handleFileOptions(n.l)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  List for saving FTP-connections
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
NewList connections.connection()

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  The menu
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Procedure mainMenu(List connections.connection())
  Repeat

    ClearConsole()
    PrintN("sFTPc-Main Menu//")
    PrintN("1 - Open FTP Connection")
    PrintN("2 - Close FTP Connection")
    PrintN("3 - File-Manager")
    PrintN("4 - Quit")
    PrintN("")

    ;; Print opened connections
    If ListSize(connections()) > 0
```

```
      FirstElement(connections())
      ForEach connections()
        PrintN("Connection number: "+Str(connections()\id))
        PrintN("Adress: "+connections()\adress)
        PrintN("Username: "+connections()\user)
        Print("Connection: ")
        checkConnection(connections()\id)
        PrintN("")
      Next
    Else
      PrintN("There are currenty no opened connections.")
    EndIf

    Print("Choose: ")

    Select Val(Input())
      Case 1
        establishConnection(connections())
      Case 2
        closeConnection(connections())
      Case 3
        ClearConsole()
        Print("Connection number: ")
        n = Val(Input())
        If IsFTP(n)
          fileManager(n)
        Else
          PrintN("Not a valid connection!")
          Print("Push enter.")
          Input()
          Continue
        EndIf
      Case 4
        End
      Default
        PrintN("Not a valid option!")
        Print("Push enter.")
        Input()
        Continue
    EndSelect
  ForEver
EndProcedure

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Procedure to open a new connection
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Procedure establishConnection(List connections.connection())
  LastElement(connections())

  If Not ListSize(connections()) = 0
    lastID = connections()\id
  Else
    lastID = 0
  EndIf

  *newElement.connection = AddElement(connections())

  ClearConsole()
  *newElement\id = lastID + 1
  Print("URL or IP: ")
  *newElement\adress = Input()
  Print("Username: ")
  *newElement\user = Input()
  Print("Password: ")
```

```
  *newElement\pass = Input()
  Print("Port (default is 21): ")
  *newElement\port = Val(Input())

  If *newElement\port = 0
    *newElement\port = 21
  EndIf

  ;; Check if the connection can get opened
  If 0 =
OpenFTP(*newElement\id,*newElement\adress,*newElement\user,*newElement\pass)
    PrintN("Could not open FTP-connection!")
    Print("Push enter.")
    Input()
    ;; Couldn't get opened? Delete the element, it's trash
    LastElement(connections())
    DeleteElement(connections())
  EndIf

  ProcedureReturn
EndProcedure

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Check the state of opened connections
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Procedure checkConnection(n.l)
  If 0 = CheckFTPConnection(n)
    PrintN("Disconnected!")
  Else
    PrintN("OK!")
  EndIf
  ProcedureReturn
EndProcedure

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Close a specific connection
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Procedure closeConnection(List connections.connection())
  ClearConsole()
  ;; Print current connections
  If ListSize(connections()) > 0
    FirstElement(connections())
    ForEach connections()
      PrintN("Connection number: "+Str(connections()\id))
      PrintN("Adress: "+connections()\adress)
      PrintN("Username: "+connections()\user)
      PrintN("")
    Next
  Else
    PrintN("There are currenty no opened connections.")
    Print("Push enter.")
    Input()
    ProcedureReturn
  EndIf

  Print("Choose Connection to close: ")
  n = Val(Input())

  If n <= 0 Or n > ListSize(connections()) Or (Not IsFTP(n))
    PrintN("Not a valid value!")
    Print("Push enter.")
    Input()
```

```
    ProcedureReturn
  EndIf

  CloseFTP(n)
  ForEach connections()
    If connections()\id = n
      Break
    EndIf
  Next

  DeleteElement(connections(),1)
  ProcedureReturn
EndProcedure

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  The build-in fileManager
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Procedure fileManager(n.l)
  Repeat
    ClearConsole()

    showDir(n)

    PrintN("")
    PrintN("Type help for options.")
    Print("> ")

    If 1 = handleFileOptions(n)
      ProcedureReturn
    EndIf
  ForEver
EndProcedure

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Show current directory
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Procedure showDir(n.l)
PrintN("Current directory: "+GetFTPDirectory(n))
  ExamineFTPDirectory(n)

  ;; List files
  While NextFTPDirectoryEntry(n)
    attributes = FTPDirectoryEntryAttributes(n)
    If attributes & #PB_FTP_ReadUser
      Print("r")
    Else
      Print("-")
    EndIf
    If attributes & #PB_FTP_WriteUser
      Print("w")
    Else
      Print("-")
    EndIf
    If attributes & #PB_FTP_ExecuteUser
      Print("x")
    Else
      Print("-")
    EndIf
    If attributes & #PB_FTP_ReadGroup
      Print("r")
    Else
      Print("-")
```

```
    EndIf
    If attributes & #PB_FTP_WriteGroup
      Print("w")
    Else
      Print("-")
    EndIf
    If attributes & #PB_FTP_ExecuteGroup
      Print("x")
    Else
      Print("-")
    EndIf
    If attributes & #PB_FTP_ReadAll
      Print("r")
    Else
      Print("-")
    EndIf
    If attributes & #PB_FTP_WriteAll
      Print("w")
    Else
      Print("-")
    EndIf
    If attributes & #PB_FTP_ExecuteAll
      Print("x")
    Else
      Print("-")
    EndIf
    Print(" "+FTPDirectoryEntryName(n))
    If FTPDirectoryEntryType(n) = #PB_FTP_File
      PrintN(" File")
    ElseIf FTPDirectoryEntryType(n) = #PB_FTP_Directory
      PrintN(" Dir")
    Else
      PrintN("")
    EndIf
  Wend

  ProcedureReturn
EndProcedure

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Handle the options of the fm
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Procedure handleFileOptions(n.l)
  option.s = Input()
  Select option
    Case "changeDir"
      Print("Dir: ")
      newDir.s = Input()
      SetFTPDirectory(n, newDir)
      ProcedureReturn 0

    Case "createDir"
      Print("Name: ")
      name.s = Input()
      CreateFTPDirectory(n, name)
      ProcedureReturn 0

    Case "deleteDir"
      Print("Name: ")
      name.s = Input()
      DeleteFTPDirectory(n, name)
      ProcedureReturn 0

    Case "delete"
```

```
      Print("Name: ")
      name.s = Input()
      DeleteFTPFile(n, name)
      ProcedureReturn 0

    Case "download"
      Print("Name: ")
      name.s = Input()
      Print("Name for saving: ")
      saveName.s = Input()
      ReceiveFTPFile(n, name, saveName)
      ProcedureReturn 0

    Case "exit"
      ProcedureReturn 1

    Case "help"
      PrintN("changeDir -> change directory")
      PrintN("createDir -> create directory")
      PrintN("deleteDir -> delete EMPTY directory")
      PrintN("delete -> delete file")
      PrintN("download -> download file")
      PrintN("exit -> return to main menu")
      PrintN("help -> this help")
      PrintN("rename -> rename file or directory")
      PrintN("send -> send file to server")
      Print("Push enter.")
      Input()
      ProcedureReturn 0

    Case "rename"
      Print("Old Name: ")
      name.s = Input()
      Print("New name: ")
      newName.s = Input()
      RenameFTPFile(n, name, newName)
      ProcedureReturn 0

    Case "send"
      Print("File: ")
      name.s = Input()
      Print("Name for saving: ")
      saveName.s = Input()
      SendFTPFile(n, name, saveName)
      ProcedureReturn 0

    Default
      PrintN("Not a valid option")
      Print("Push enter.")
      Input()
      ProcedureReturn 0

  EndSelect
EndProcedure

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Start the program
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
OpenConsole()
EnableGraphicalConsole(1)
mainMenu(connections())
```

6.1.1 Was soll das Programm tun?

Dieses Programm stellt einen einfachen, konsolenbasierten FTP-Client dar. Wenn das Programm gestartet wird, sieht man im Terminal ein Menü. Über den ersten Menüpunkt kann man eine FTP-Session eröffnen, über den zweiten kann man sie wieder schließen und über den dritten kann man die Dateien auf dem FTP-Server durchsuchen und bearbeiten. Der vierte schließt das Programm wieder. Das Programm ist dabei in der Lage mehrere FTP-Sessions zu verwalten. Man muss immer angeben, welche Session gerade bearbeitet werden soll.

6.1.2 Wie funktioniert das Programm?

Am Anfang wird über *InitNetwork()* die Netzwerk-Bibliothek geladen, wodurch die Befehle für die FTP-Verwaltung verfügbar werden. Die Struktur ist da, um die FTP-Verbindungen zu verwalten. Danach folgt die Deklaration mehrerer Verbindungen, sowie die Deklaration einer Liste, in der die FTP-Verbindungen, repräsentiert über die Struktur *connection*, organisiert werden. Danach werden die Prozeduren definiert. Ganz am Ende des Programmes wird die Konsole geöffnet. *EnableGraphicalConsole(1)* ermöglicht das Löschen des Inhalts der Konsole. Der letzte Befehl ruft das Hauptmenü auf, dass in der ersten Prozedur definiert ist.

- *mainMenu()*: Die Prozedur erwartet, dass man ihr die Liste mit den Verbindungen übergibt. Es wird lediglich das Menü dargestellt und auf Eingaben mittels Fallunterscheidungen reagiert. Wenn FTP-Verbindungen eröffnet wurden, stellt das Programm diese dar. Beim Schließen des Programms muss nicht darauf geachtet werden, dass alle Verbindungen geschlossen werden, dies wird automatisch beim Schließen erledigt.
- *establishConnection()*: Diese Prozedur erwartet ebenfalls die Liste der Verbindungen. Zuerst wird die Nummer der letzten FTP-Verbindung herausgefunden. Das ist wichtig für die Vergabe einer Nummer an die neue Verbindung. Danach wird ein neues Element zur Liste hinzugefügt. *AddElement()* gibt die Adresse des neuen Elements zurück, die im Zeiger gespeichert wird. Über den Zeiger wird danach auf das Element zugegriffen und die Daten werden eingegeben. Mit *OpenFTP()* wird dann versucht die Verbindung herzustellen. Wenn sie nicht hergestellt werden kann, wird sie gelöscht.
- *checkConnection()*: Diese Prozedur überprüft lediglich, ob die Verbindung noch besteht. Dies wird im Hauptmenü genutzt, wo dies angezeigt wird.
- *closeConnection()*: Es werden alle geöffneten Verbindungen angezeigt. Man wählt eine aus und diese wird geschlossen und aus der Liste gelöscht. Es wird dabei überprüft, ob es sich wirklich um eine geöffnete Verbindung handelt.
- *fileManager()*: Diese Prozedur steuert den Dateimanager. Es wird das aktuell ausgewählte Verzeichnis angezeigt und man kann mit diesem interagieren.
- *showDir()*: Es wird einfach der Inhalt des Verzeichnis aufgelistet, zusammen mit den Rechten des Benutzers, mit dem die Verbindung eröffnet wurde.
- *handleFileOptions()*: Diese Prozedur wird aus *fileManager()* aufgerufen. Man kann verschiedene Befehle eingeben. Welche das sind, wird über den Befehl *help* angezeigt.

6.2 reversePolishNotation

```
OpenConsole()

NewList entries.s()
PrintN("Please type only numbers and arithmetic operators(+,-,*,/)")
PrintN("Input:")

Repeat
  entry.s = Input()
  AddElement(entries())
  entries() = entry

  If entries() = "="
    DeleteElement(entries())
    PrintN(entries())
    Input()
    Break
  EndIf
  If entries() = "-" Or entries() = "+" Or entries() = "/" Or entries() = "*"
    If ListSize(entries()) < 3
      PrintN("Not enough Elements for calculating")
      Input()
      Break
    EndIf

    index = ListIndex(entries())
    SelectElement(entries(),index-2)

    a.f = ValF(entries())
    DeleteElement(entries())
    NextElement(entries())

    b.f = ValF(entries())
    DeleteElement(entries())
    NextElement(entries())

    Select entries()
      Case "-"
        InsertElement(entries())
        entries() = StrF(a-b)
        NextElement(entries())
        DeleteElement(entries())
      Case "+"
        InsertElement(entries())
        entries() = StrF(a+b)
        NextElement(entries())
        DeleteElement(entries())
      Case "/"
        InsertElement(entries())
        entries() = StrF(a/b)
        NextElement(entries())
        DeleteElement(entries())
      Case "*"
        InsertElement(entries())
        entries() = StrF(a*b)
        NextElement(entries())
        DeleteElement(entries())
    EndSelect
  EndIf
ForEver
```

6.2.1 Was soll das Programm tun?

Das Programm stellt einen Taschenrechner dar, der jedoch mit der umgekehrten polnischen Notation arbeitet. Bei dieser schreibt man die zwei Zahlen, mit denen gerechnet werden soll zuerst und danach die Rechenoperation. "2+8" würde man also als "2 8 +" schreiben. Hierdurch kann man sich die Klammersetzung sparen: "2*(5-3)" könnte man schreiben als "2 5 3 - *".

6.2.2 Wie funktioniert das Programm?

Das Programm ist sehr einfach gehalten. Es werden alle Rechenzeichen und Zahlen hintereinander eingegeben. Jede Eingabe wird dabei zu einer Liste hinzugefügt. Sobald ein Rechenzeichen eingeben wird, wird entsprechend auf dieses reagiert, wobei mit den beiden Zahlen gerechnet wird, die zuvor zur Liste hinzugefügt wurden. Nach dem Beispiel aus dem vorigen Abschnitt passiert also folgendes: Es werden hintereinander die Zahlen 2, 5 und 3 eingegeben. Nun wird das Minus eingeben, woraufhin die Zahlen 5 und 3 aus der Liste geholt und danach subtrahiert werden. In der Liste sind also nun die Zahlen 2 und 2. Gibt man nun die Multiplikation ein, geschieht das gleiche mit den beiden Zweien, sodass in der Liste nur noch eine 4 ist. Mit "=" wird diese 4 ausgegeben.

7 Autoren

Edits	User
1	Dirk Huenniger[1]
10	Hoermi t[2]
210	Raymontag[3]

1 http://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger
2 http://de.wikibooks.org/wiki/Benutzer:Hoermi_t
3 http://de.wikibooks.org/wiki/Benutzer:Raymontag

Abbildungsverzeichnis

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. http://artlibre.org/licence/lal/de
- CFR: Copyright free use.

- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[4]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

4 Kapitel 8 auf Seite 81

1	Raymontag	
2	Raymontag	
3	Raymontag	
4	Raymontag	

8 Licenses

8.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”. * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.

A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from

conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

8.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "freeïn the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Versionöf the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A SSecondary Sectionïs a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The Ïnvariant Sectionsäre certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Textsäre certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent"copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparentïs called Öpaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page"means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher"means any person or entity that distributes copies of the Document to the public.

A section Ëntitled XYZ"means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as Äcknowledgements", "Dedications", Ëndorsements", or "History".) To "Preserve the Titleöf such a section when you modify the Document means that it remains a section Ëntitled XYZäccording to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "Historyïn the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "Historyßection. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled Äcknowledgementsör "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled Ëndorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled Ëndorsementsör to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled Ëndorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "Historyïn the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled Äcknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled Ëndorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ãggregateïf the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled Äcknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ör any later versionäpplies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site"(or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration"(or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA"means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

Ïncorporate"means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is ëligible for relicensingïf it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts."line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

8.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.