Projekt c/ | Vorwort

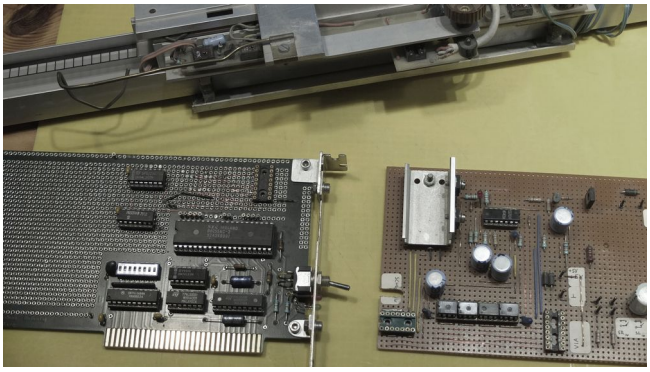
Inhaltsverzeichnis

Vorwort

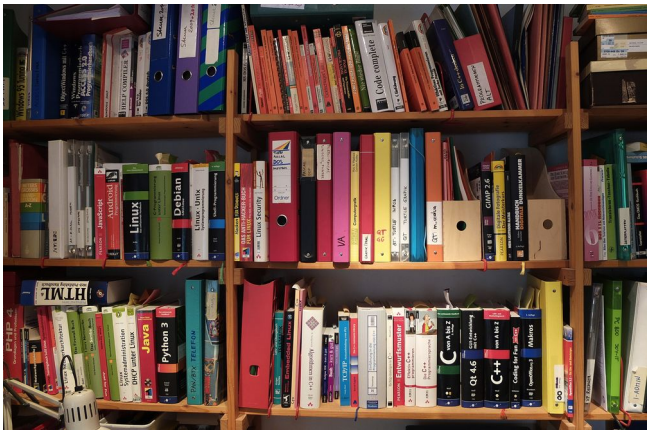
Rückblick

Das Programmieren begleitet mich schon seit über 50 Jahren. Mein erstes Programm habe an der Technischen Universität Graz damals noch auf Lochkarte codiert. Programmieren war und ist für mich nur ein Hilfsmittel, Neues zu erfinden. Mit meinem ersten Apple II Nachbau haben sich erstmalig ungeahnte Möglichkeiten eröffnet, Neues zu entwickeln.

Das erste herzeigbare Projekt war eine Steuerung für die Strickmaschine meiner Frau. Dazu gab es Hilfsprogramme zum Erstellen der zweifarbigen Mustervorlagen und zum Zeichnen von Schnitten. Realisiert wurde das Projekt auf einem Apple II mit selbstgebautem Apple-Interface und Positionsgeber. Die Software war eine Mischung von USCD-Pascal auf einer CPM Erweiterungskarte und Maschinensprache für Prozessor 6202. Die Entwicklung erfolgte ohne irgendeine Debuggersoftware. Die Echtzeitroutinen wurden alle vollständig am Papier entworfen. Die praktischen Tests ohne Debugger lieferten dann nur die Rückmeldung: Funktioniert oder funktioniert nicht.



Die Pionierzeit war geprägt durch die Entwicklung neuer Programmiersprachen und deren ausführlicher Diskussion. Aus heutiger Sicht wurde dabei aber übersehen, dass die Programmiersprachen selbst nur einen kleinen Teil einer vollständigen Entwicklungsumgebung ausmachen.



Im Laufe der Zeit habe ich mich daher mit vielen Programmiersprachen wie Fortran, Basic, Maschinensprachen, Ada, Lisp, Logo, Pascal, C, C++, Access-Basic. Html und PHP usw. beschäftigt. Dabei entstanden viele konkrete Problemlösungen für mich und für andere User.

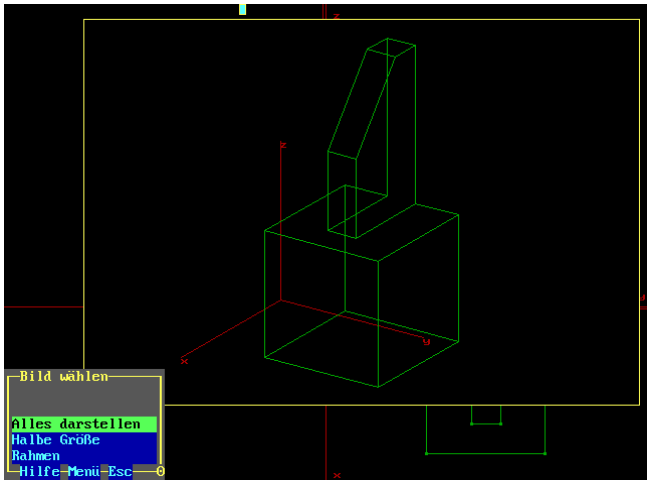
Eine durchgehende Erfahrung beim Entwickeln neuer Projekte war, dass in den verschiedenen Programmierumgebungen immer wieder dieselben Funktionen neu codiert werden mussten.

Aus der Vielzahl von Projekten möchte ich im Folgenden drei komplexere Beispiele vorstellen.

Beispiel 1:

- ▷ Für den Unterricht in 'Geometrisch Zeichnen' habe ich ein echtes 3D CAD-Programm namens DGZ entwickelt. Es lief unter Dos und Windows. Das selbst programmierte graphische Fenster-System lief direkt auf der Grafikhardware.

  Es wurde vom Bundesministerium angekauft und in Österreich flächendeckend in den Schulstufen 6 und 7 (absturzfrei!) verwendet. Die Entwicklungszeit betrug ca. 3 Mannjahre.



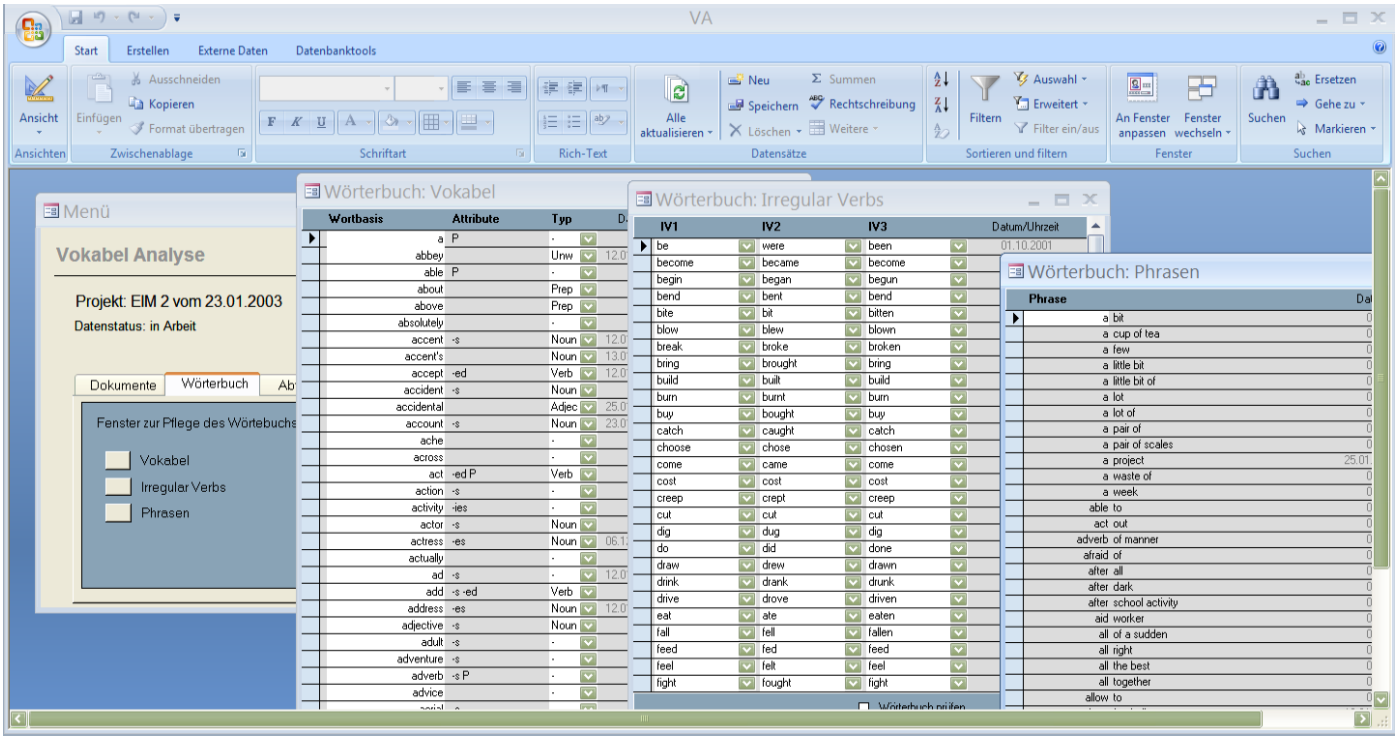
Beispiel 2:

▷ Ein Vokabel-Analyse-Programm für Englisch Lehrbücher der Autoren Dr. Herbert Puchta und Team. Das Programm stellte sicher, dass die Vokabel mehrerer Lehrbücher nicht vor ihrer Einführung verwendet wurden.

Die Texte mehrerer Bücher wurden automatisch gescannt und die Vokabel und Phrasen in einer Datenbank erfasst.

Die verschiedenen Endungen und Formen der Vokabel mussten nur einmal manuell zugeordnet werden.

Die Analyse dieser Datenbasis erwies sich als sehr komplex. Für die ersten Versionen noch unter Dos musste ich eigene Datenspeicher-Funktionen schreiben. Spätere Version verwendeten als Datenbank Access.



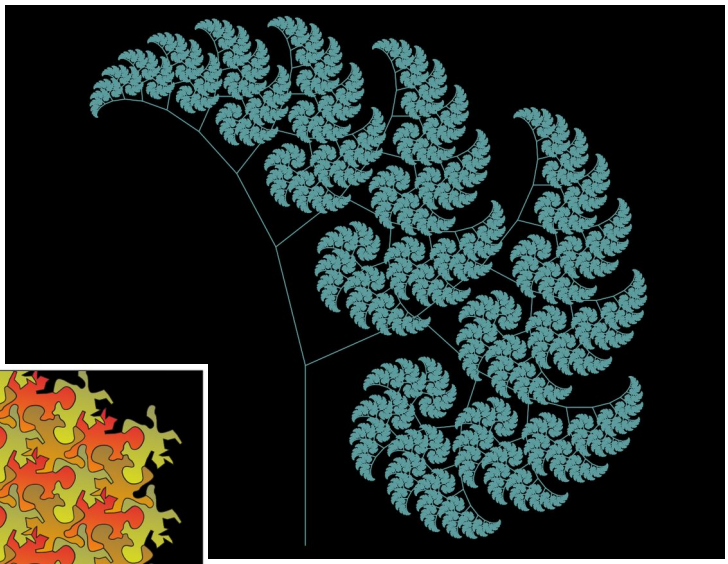
Beispiel 3:

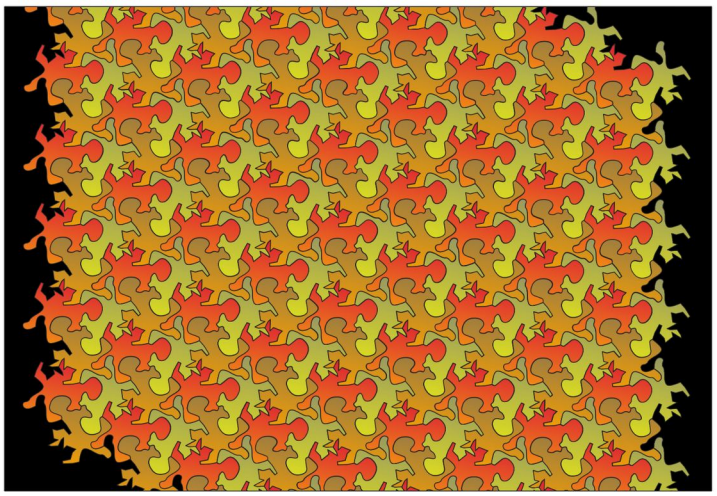
▷ Ein Funktionsplotter unter C++. Mit dem Programm können mathematisch definierte Computergrafiken für den Bildschirm und Ausdruck berechnet werden.

Die Grafiken können im Programm oder über Scriptdateien erstellt werden. Für Kurven gibt es einen Spline-Editor.

Die Auswahl der Zeichenflächen müsste verbessert werden. Durch die mehrfache Vererbung wäre das aber jetzt ein mühsames Unterfangen.

Mein Hauptproblem ist es aber, Qt und das Projekt Funktionsplotter auf neuen Systemen zum Laufen zu bringen.





Weitere Bilder finde man auf http://www.schmuckhexen.at/programs/turtlepdfs/index.html

Der Anfang von Projekt c/

Vor ungefähr 10 Jahren habe ich begonnen, mich mit Steuerungen am Raspberry Pi und auf Arduino zu beschäftigen. Dazu habe ich nach einer für mich passenden Programmierumgebung gesucht. Bei der Auswahl sind meine langjährigen Erfahrungen eingeflossen.

Meine Anforderungen an eine Entwicklungsumgebung:

- ▷ Komplexe Programme auf schwacher Hardware sicher und zügig zu entwickeln.
- ▷ Die Programme sollten langlebig und auch später gut zu erweitern sein.
- ▷ Alle Lösungen sollten unabhängig von anderen Personen, Firmen und Lizenzen sein.
- ▷ Updates der Betriebssysteme sollen die Funktion der Programme nicht stören.
- ▷ Die Lösungen sollen auf verschiedenen Rechnern sofort laufen.
- ▷ Bereits vorhandene Lösungen oder eigene Entwicklungen sollten einfach integriert werden können.
- ▷ Gutes Fehlermanagement während der Entwicklung und einfache Fehlersuche in laufenden Programmen.
- ▷ Leichte Erweiterung/Verbesserung von fertigen Programmen.

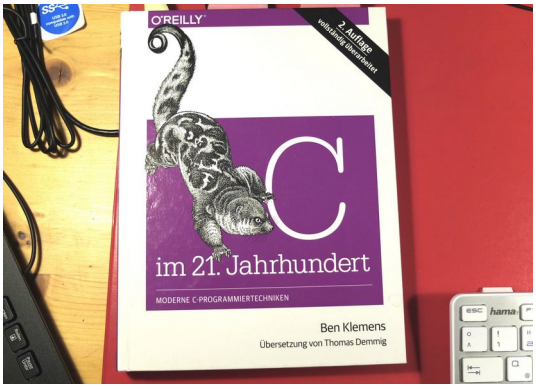
Das Betriebssystem Linux und der C-Compiler mit den Laufzeitbibliotheken Libc waren geeignete Kandidaten. Auf Grund meiner früheren Erfahrungen war die Sprache C für mich zunächst aber kein Wunschkandidat.

Im Laufe von etwa 10 Jahren ist dann das 'Projekt /c' entstanden und die Sprache C hat meine Erwartungen mehr als übertroffen. Das liegt wohl daran, dass bei umfangreichen Projekten die Bedeutung des reinen C Sprachanteils eher gering ist. Es geht um komplexe Projekt-Organisation, die Verfügbarkeit von externen Lösungen, die Wiederverwertung von Programmteilen, die Dokumentation, die Fehlererkennung, die Langzeit-Stabilität und noch weitere Anforderungen.

Die Sprache C lässt sich schon auf 120 Seiten vollständig beschreiben.

Die Sprachkonstrukte wirken aus theoretischer Sicht oft eigenartig.
Bei der praktischen Anwendung zeigen sich aber die Stärken von C.

Für fortgeschrittene Programmierung gibt es gute weiterführende Bücher.

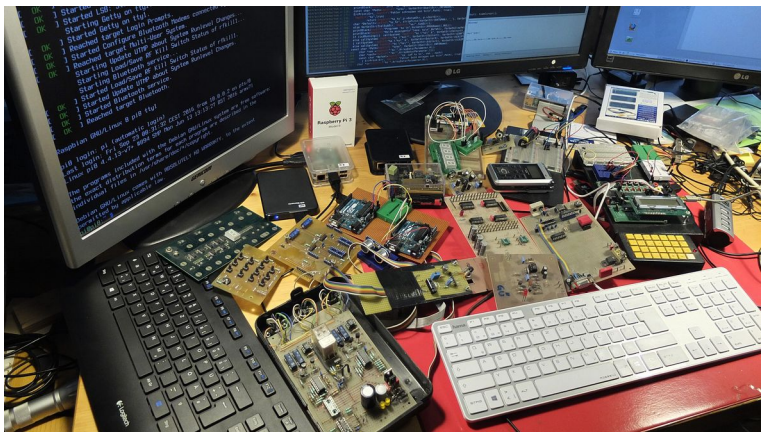


C im 21. Jahrhundert
by Ben Klemens
Publisher(s): O'Reilly Verlag

> Aus dem Vorwort:
>
> **C ist Punkrock**
>
> C besitzt eine Handvoll Schlüsselwörter, es hat seine Ecken und Kanten - und es rockt. ... Sie können die grundlegenden Funktionen ziemlich rasch erlernen und dann den Rest Ihres Lebens damit verbringen, besser zu werden. ...

Leider fehlen Bücher, die die notwendige Projektorganisation unter C gut beschreiben! Die Organisation von C Projekten ist nicht aufwändig, aber es gibt viele Möglichkeiten. Diese lassen sich sicher auf wenigen Seiten beschreiben, es ist aber sehr mühsam, sich das alles selbst zusammenzusuchen.

Mein Arbeitsplatz:



Projekt c/

Die Werkstatt

Die Werkstatt Projekt c/ ist ohne Installation auf jedem Linux-System mit gcc Compiler sofort lauffähig. Es gib keine speziellen Systemvoraussetzungen oder Abhängigkeiten von anderen Programmen. Ein make-Aufruf kann alle Programme erzeugen.

Die Grundidee ist es, die in den Linuxbibliotheken vorhandenen C-Funktionen einfach und sicher verfügbar zu machen. Dazu zwei Beispiele mit nicht trivialen Linuxfunktionen.

Beispiel 1:

Mit Werkstatt Projekt c/ erstellte Programme warten in getTaste(Prompt) auf Tasteneingaben oder Interrupts. Intern basieren alle Eingabefunktionen auf readKbd().

```
clrScr();                           // Bibliothek utils.h bereitstellen und Bildschirm löschen
...

switch(getTaste( BEFEHLEsc ))       // Das Programm wartet auf eine Tasteneingabe oder Interrupt
{ case 'r': runFunktion(); break;   // Funktionsaufruf bei Taste 'r'

  case taste_select:                // Beispiel: Sensor-Interrupt wurde ausgelöst
    ...                             // Event-Handler wurde angehalten
  break;

  case taste_esc: return;           // Ende mit ESC-Taste
}
```

Mit clrScr() bindet der Compiler die Funktion getTaste() ins Programm ein. In allen Eingabefunktionen können bei Bedarf Interrupts über den select() Mechanismus aktiviert werden. Siehe: www.schmuckhexen.at/programs/c/clar_objekte2.pdf

Beispiel 2:

Dialog zur File- oder Ordnerwahl aus Bibliothek c/lib/iocon. Die verwendeten Linuxfunktionen sind sehr komplex.

```
void TestRunBoxDirWahl()
{
   DirFilterSetWildcardsStr("*.jpg,*.pnp", ','); // Wildcardstring, Separatozeichen
   DirFilterSetFlags(FNM_Flags);                 // Siehe man fnmatch(). Z.B. FNM_CASEFOLD für Wildcards case-insensitively.

   const char *erg=runBoxDirWahl
   ( 1,1,-1,0,       // Anzeige: Zeile, Spalte, Höhe oder <1 vom Bildschirmende, Breite oder <1 vom Zeilenende
     StartPath,
     DirFilterFiles, // Filter: Ordner, "../", sichtbare Files und Links anzeigen.
                     // Verwendet DirFilter Wildcards und DirFilterFlags
     RetTyp,         // Rückgabe festlegen: Ordner, Files,  Ordner und Files, (keine) hasAccess() Prüfung durchführen
     FCap4,          // Titelfarbe: NULL Defaultfarbe oder Farbe
     "Titeltext",    // NULL oder Zeilen mit /n getrennt
     OrdnerDlg       // NULL oder externen Ordnerdialog verwenden
   );

  printLn();printLine(0); clrEos();     // Leerzeile, Trennlinie, bis zum Ende des Bildschirms löschen

  printf("Ergebnis: "Fy"%s"FN"\n",erg); // Ergebis in Farbe Yellow (Fy) anzeigen
```

Siehe: www.schmuckhexen.at/programs/c/clar_objekte2.pdf

Das Basiskonzept

Die Werkstatt Projekt c/ ist ohne Installation auf jedem Linux-System mit gcc Compiler sofort lauffähig. Es gib keine speziellen Systemvoraussetzungen oder externe Abhängigkeiten. Ein make-Aufruf kann alle Programme erzeugen.

Die Lösung:

- ▷ Alle Dateien befinden sich in einem Ordner. Dieser kann ohne Installation verschoben oder auf andere Systeme kopiert werden.
  Parallel zum eigentlichen Projektordner können weitere Programmordner - z.B. für Testprogramme - angelegt werden.
- ▷ Die Systemvoraussetzungen Linux, C-Compiler gcc und die Bibliothek Libc sind immer vorhanden.
- ▷ Übersichtliche Ausgaben auf der Console oder in Terminals auf grafischen Oberflächen.
- ▷ Komfortable Eingabefunktion im Terminal.
- ▷ Alles kann über ssh oder ssh -X benutzt werden.
- ▷ Keine Lizenzen oder Abhängigkeiten von externen Bibliotheken.

Die einfache Entwicklung und Wartung von komplexen Projekten wird durch die Stärken der C/C++ Familie unterstützt:

- ▷ C ist auf jedem Linuxsystem seit Jahrzehnten verfügbar. Der Compiler gcc und Libc sind sehr langzeitstabil.
  Daher laufen die Programme von Projekt c/ seit 10 Jahren auf allen Versionen von gcc und Libc.
- ▷ Die Objekthierarchien bleiben unter C immer sehr flach. Daher können auch sehr komplexe Projekte
  während der Entwicklung leicht umorganisiert werden.
  Fertige Projekte können leicht weiter entwickelt werden oder in neue Projekte übernommen werden.
  Der relevante C-Code ist auch noch nach Jahren im Programmcode leicht zu finden und zu ändern.
  Fehler sind leicht zu beheben.
- ▷ Objektorientierte Programmierung ist über Module auch in C sehr gut umsetzbar.
- ▷ C ist auf Wunsch auch sehr systemnah. Es gibt für alles schon gute Lösungen.
- ▷ Viele Funktionen können einfach und stabil auch über Bash-Befehle implementiert werden.
- ▷ Make verwaltet sehr inhomogene Projekt-Sourcen von PC, Raspberry Pi, Arduino und anderen Systemen sehr gut.
- ▷ Compilieren und Testen geht sehr schnell - auch über ssh -X Verbindungen.
- ▷ Copy & Paste von Funktionsblöcken ist sehr einfach und sicher.
- ▷ Mit Vorlagen wird das Erstellen neuer Programme einfach.

Werkzeuge

Alle Projekt c/ angebotenen Funktionen oder Werkzeuge **können** aber **müssen** nicht verwendet werden!

- ▷ Statische Bibliotheksfunktionen für die in jedem Projekt immer wiederkehrenden Funktionen.
  Das sichere Funktionieren dieser Funktionen wurde mit umfangreichen Testprogrammen unter c/libtest/ geprüft.
  Die Konsistenz von etwaigen Änderungen kann mit diesen Testprogrammen sicher verifiziert werden.
- ▷ Ein universal makefile: Gut dokumentiert und mit bedingter Compilierung für Pi oder PC.
- ▷ Automatische Erstellung von neuen Programmgerüsten aus eigenen Vorlagen.
- ▷ Kommunikation zwischen PC oder Raspberry und Arduino über USB.
- ▷ Fernsteuerungen mit ssh oder ssh -X.

Die Bibliotheksfunktionen können sofort verwendet werden, da die erforderlichen Objekte immer automatisch angelegt werden:

- ▷ Automatische Verwaltung von temporären Strings. Sie benötigen kein malloc() und free().
  Alle Funktionen mit String-Rückgabe liefern einheitlich diese temporäre Strings.
- ▷ Umfangreiche Stringfunktionen. Die Funktionen aus Libc wurden überschrieben und vertragen nun auch 0 und NULL Strings.
- ▷ Objekt Var implementiert globale Laufzeitvariblen. Sie werden bei Verwendung automatisch erzeugt und können
  auch aus Konfigurationsdateien direkt geladen oder nach Konfigurationsdateien geschrieben werden.
- ▷ Einfache, einheitliche, aber reichhaltige Fehlerbehandlung mit Err-Objekt.
- ▷ Die Speicherfreigabe freePtr() ersetzt free(). freePtr() kann mit NULL, Heap, Text und Stackpointer verwendet werden!
- ▷ Sicheres und schnelles Array-Objekt für String und beliebige Strukturen.
  Die Strukturen können auch in Dateien gespeichert/gelesen werden.
- ▷ Fortlaufende Ausgaben können sehr einfach erzeugt und mit einer less-Funktion gescrollt werden.
- ▷ Funktionen zum Lesen und Schreiben von C-artigen Konfigurations-, Script- oder Datendateien.
- ▷ usw.

Input/Output Funktionen

Die von Projekt c/ angebotenen Input/Output Funktionen **können** aber **müssen** nicht verwendet werden!

Durch die Bibliotheksfunktionen werden praktische Input/Output Funktionen für C bereitgestellt:

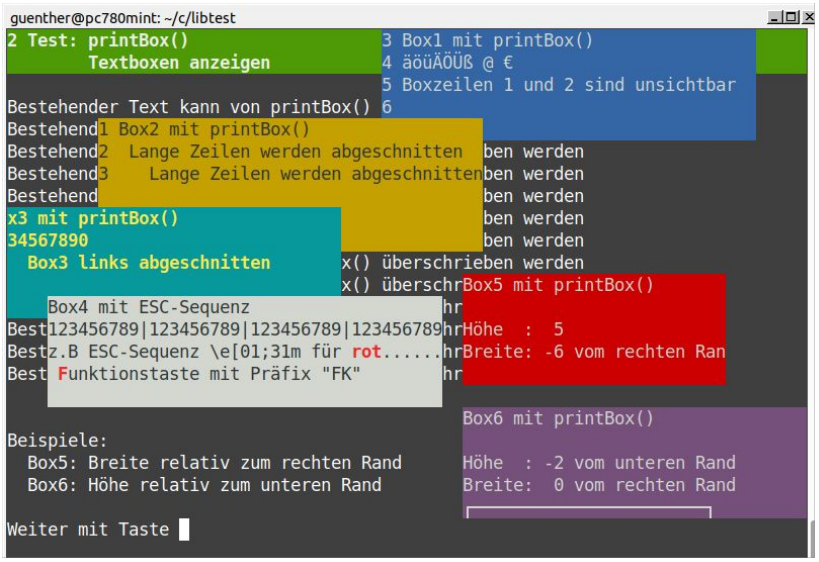
Input

- ▷ Inputfunktionen: bieten blockierendes oder nichtblockierendes IO-Multiplexing an.
- ▷ Low Level Input: peekTaste(), chkTaste(), getTaste(). Sie können für die Interrupt-Behandlung die Taste taste_select liefern.
- ▷ High Level Input: Eingabezeile mit der üblichen Cursorsteuerung, Defaultwerten, Abbruch und Fehlerprüfung.
  readLn(), readDouble(), readInt32(), readUInt32(), readInt16(), readUInt16()
  readHex16(), readHex32()
  readUInt16() und readHex arbeiten auch mit Hex- (0x...) und Binäreingaben (0b...).

Output

- ▷ Ausgabe fortlaufend: printf(), printLine(). printBlock() kann farbige Textzeilen in Terminalbreite ausgeben.
- ▷ Ausgabe positioniert: printBox() kann farbigen Text in einem Rechteck an fixen Bildschirmpositionen ausgeben.
  Der Box-Text wird am Rechteck geclipped und das Rechteck wird am Terminalfenster geclippt.
  printBox() ist das Basisobjekt für alle Dialogboxen.
- ▷ Input/Output Dialoge: Menu, Dateidialog, Listwahl, usw.

Positionierte Ausgaben mit printBox():



Systemaufrufe

Die Bibliotheksfunktionen bieten sichere und komplexe Systemaufrufe. Sie **können** aber **müssen** nicht verwendet werden!

- ▷ Parameterprüfung für alle Aufrufe. NULL, 0, "" oder NIL sind gültige Parameter und führen nie zum Absturz!
- ▷ Einheitliche und flexible Fehlerbehandlung mit Err-Objekt. Automatische Umleitung von Fehleranzeigen in eine Logdatei.
- ▷ Aufruf von anderen Programmen mit Auswertung der Rückgaben in Form einer Zeile oder Liste.
- ▷ Eine laufende Programminstanz kann vollständig durch ein anderes Programm ersetzt werden.
  Damit lassen sich komplexe Projekte einfach in überschaubare Programmmodule zerlegen.
- ▷ Einfache Funktionsaufrufe für komplexe Linux-Funktionen. Beispiele:

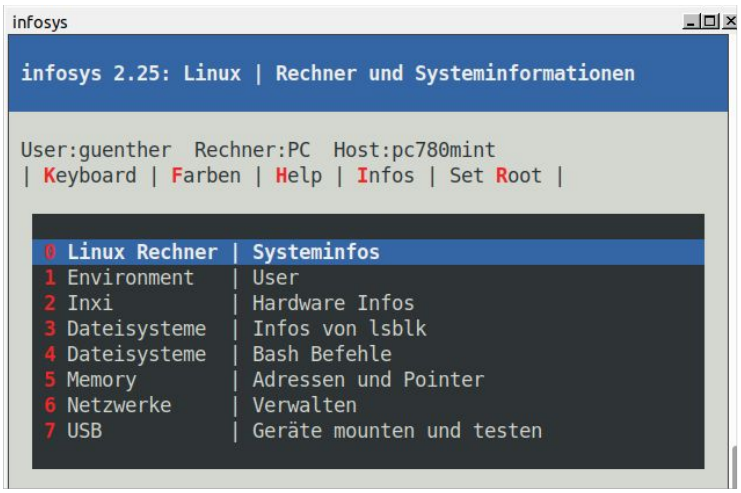
ScanDir() :	Gefilterte Dateilisten mit Wildcards von Verzeichnissen. Verwendet die interne Funktion scandir().
ScanFile() :	Gefilterte Dateilisten von Verzeichnisbäumen.
Datum/Zeit:	Zugriff und Umrechnungen zwischen den verschiedenen Linux Time Formaten.
openStdLog() :	Die gesamte Terminalausgabe wird in eine Logdatei umgeleitet.
getPopenArray(Befehl):	Die Funktion speichert die Rückgaben des Bash-Befehls in einem Array.
printLessOn/Off:	Fortlaufende Terminalausgaben und Fehler werden mit dem Pager less angezeigt.
usw.	

Für alle Bibliotheksfunktionen gilt:

- ▷ Die Funktionen akzeptieren immer alle möglichen Aufrufparameter.
- ▷ String-Rückgaben von Funktionen verwenden immer temporäre Strings.
- ▷ Eigene Fehler und Systemfehler werden im Error-Objekt Err gesammelt.
- ▷ Objektorientierte Implementierungen.
- ▷ Jede Funktion kann unabhängig verwendet werden.

Dialoge

Bibliotheksfunktionen bieten auch sichere und komplexe Dialogaufrufe. Sie **können** aber **müssen** nicht verwendet werden!



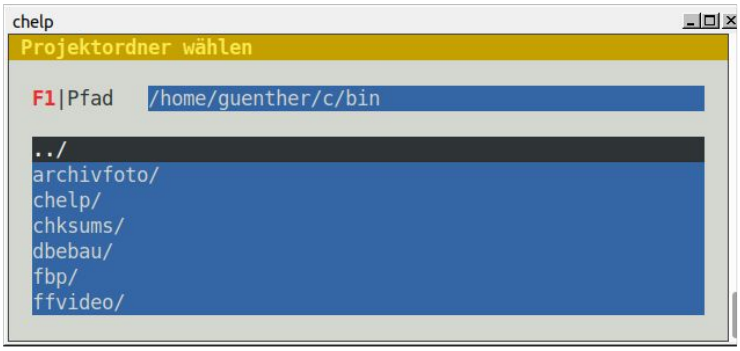
▷ **Menus**

Die Bedienung erfolgt mit Cursor- oder Funktionstasten.

Die Menus werden als normale C-Struktur im Sourcecode definiert. Sie können aber auch zur Laufzeit aus Script-Dateien geladen werden.

In den Menus können C-Funktionen, Bash-Befehle, Scripte, Funktionstasten oder andere Menus aufgerufen werden.

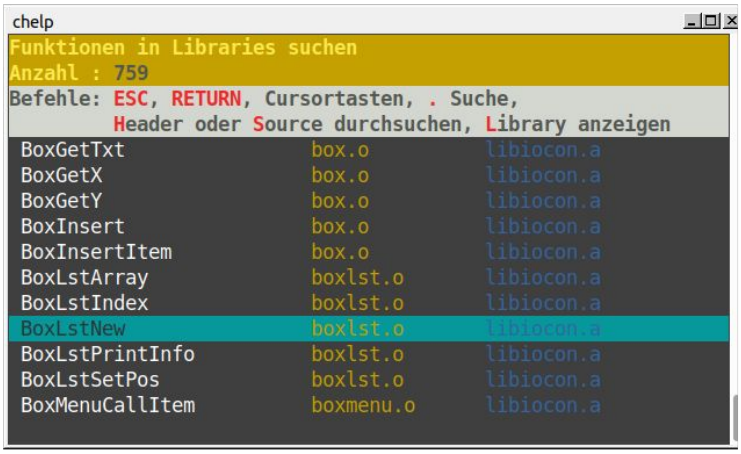
Der Aufruf runMenu(MenuDef, 0) wickelt alle Menufunktionen des Menus MenuDef automatisch ab. Die Rückgabe kann eine globale Funktionstaste sein.



▷ **Dialoge zur Datei- oder Ordnerwahl**

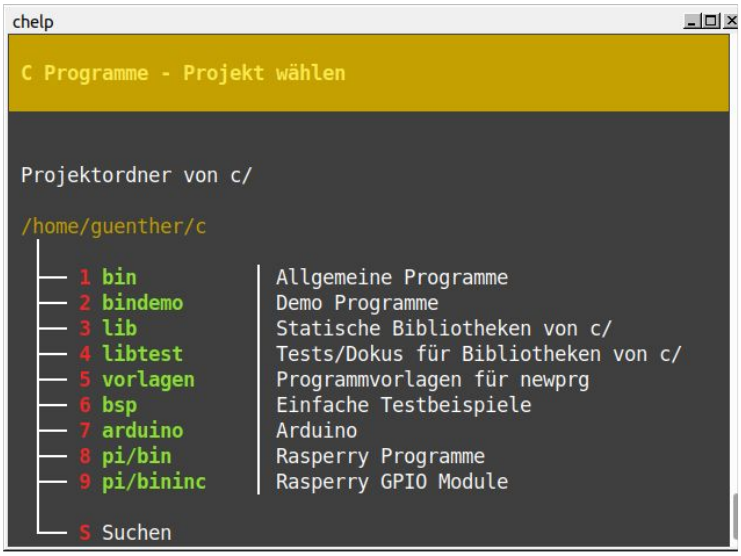
Es können Wildcards verwendet werden und es gibt umfangreiche Filtermöglichkeiten.

Hilfe mit F1



▷ **Dialog zur Auswahl aus Listen**

In den Listen kann eine Zeile mit Taste RETURN oder eine beliebige Funktionstaste gewählt werden.



▷ **Auswahlbäume**

Für System-Infos gibt es vordefinierte Auswahlbäume. Diese können aber auch im Programm zeilenweise erstellt werden.

Projekt c/ testen

Um sich einen praktischen Überblick zu verschaffen, kann Projekt c/ ohne großen Zeitaufwand getestet werden.
Danach kann der Projektordner ohne Spuren gelöscht werden.

Die Grundidee von Projekt c/ ist es, sich eine persönliche Arbeitsumgebung zu erstellen.

Mein Projekt c/ enthält sehr viele Programme, die während meiner Arbeit entstanden sind. Diese Programme sind teilweise sehr komplex und dienen nur zur Demonstration. Sie können einfach gelöscht werden. Eine genaue Datei- und Ordnerübersicht findet man in clar_start.pdf.

Kurzübersicht:

c/bsp	Optional: Probier-Programme aus der Anfangszeit von Projekt c/.
c/libtest	Optional: Testprogramme für alle Bibliotheksfunktionen zur Qualitätskontrolle.
c/lib	Die unbedingt notwendigen Bibliotheken!
c/bin	Eigene Programme für PC und Pi
c/pi	Eigene Programme nur für Raspberry Pi
c/pi/bininc	Linkbibliotheken für Raspberry Pi

Ein eigenes Projekt c/ sollte neben den Bibliotheken c/lib das Verwaltungsprogramm c/bin/chelp enthalten!
Das Programm c/bin/infosys erleichtert die Administration von Rechnern und sollte den persönlichen Bedürfnissen angepasst werden.

Neue Programme anlegen:

newprg	Mit newprg können funktionierende Programmgerüste aus Vorlagen erstellt werden.

Download

Homepage und Downloads: www.schmuckhexen.at/programms/c/index.html

Allgemeine Beschreibungen

Die allgemeinen Dokumentationen findet man unter c/1_Dokus oder im Internet:

Vorwort:	www.schmuckhexen.at/programs/c/clar_vorwort.pdf	c/1_Dokus/clar_vorwort.pdf
Projekt c/ einrichten. Die ersten Schritte:	www.schmuckhexen.at/programs/c/clar_start.pdf	c/1_Dokus/clar_start.pdf
Projekthilfe und Projektmanager:	www.schmuckhexen.at/programs/c/clar_chelp.pdf	c/1_Dokus/clar_chelp.pdf
Ein neues C Programm erstellen:	www.schmuckhexen.at/programs/c/clar_projekt.pdf	c/1_Dokus/clar_projekt.pdf
Basisobjekte ohne Terminal In/Ouput:	www.schmuckhexen.at/programs/c/clar_objekte1.pdf	c/1_Dokus/clar_objekte1.pdf
Terminalsteuerung Box-Objekte für In/Ouput:	www.schmuckhexen.at/programs/c/clar_objekte2.pdf	c/1_Dokus/clar_objekte2.pdf
Datenspeicherung:	www.schmuckhexen.at/programs/c/clar_objekte3.pdf	c/1_Dokus/clar_objekte3.pdf

C-Programme und Bibliotheken

Orientierungshinweise:

- ▷ Die ersten Infos zu den C-Programmen oder Bibliotheken findet man in der Hilfedatei '1_read.me' im jeweiligen Programmordner.
- ▷ Für aufwändige Programme gibt es Dokumentationen im Format *.odt oder *.pdf im jeweiligen Ordner ProgName/bin/_ProgName/
- ▷ Der Programmcode wurde hauptsächlich in den C-Headern der Programme oder Bibliotheken dokumentiert.
- ▷ Hilfe zu den fertigen Programmen liefert immer die Startoption '-h'.

Einstiege:

Programm chelp	Zugriff auf alle Dokus und sonstige Resourcen von Projekt c/
Projekt c/ Einstieg und Übersicht	c/1_read.me
Header/Dokus für Bibliotheksfunktionen	c/lib/1_read.me
Testprogramme für Bibliotheksfunktionen	c/libtest/1_read.me

Autoren gesucht

Linux

Die Erfolgsgeschichte von Linux und der 'GNU General Public License' basiert nicht unwesentlich auf dem C-Compiler und den Linux Bibliotheken. Die Basiswerkzeuge C-Compiler, make, libs und Bash sind aber nicht sehr beliebt. Die von mir zusammengestellte Werkstatt 'Projekt /c' kann den Einstieg in C-Projekte sehr erleichtern. Für mich war es wesentlich, die Linuxbibliotheken einfach verfügbar zu machen ohne weitere Abhängigkeiten oder einen Zwang, die angebotenen Werkzeuge auch benutzen zu müssen.

Über die Programmiersprache C selbst gibt es viele gute Bücher. Linux war aber nicht wegen der Sprache C so erfolgreich, sondern wegen des Gesamtkonzepts. Die Vorteile der Basiswerkzeuge C-Compiler, make, libs und Bash liegen in der großen Flexibilität.

Für Newcomer fehlt aber eine gute praktische Beschreibung, wie man umfangreiche und langlebige Programme baut. Die Organisation von C-Projekten ist zwar nicht aufwändig und lässt sich auch auf wenigen Seiten beschreiben. Es ist aber ist sehr mühsam, sich diese Informationen selbst zusammenzusuchen. Eine erfolgreiche Projektorganisation hat außerdem sehr wenig mit der verwendeten Programmiersprache zu tun.

Ich würde gerne im Sinne des Open Source Gedankens die Werkstatt 'Projekt /c' anderen nahebringen und suche einen Autor für ein Buch oder einen Blog rund um das Thema 'Praktischer Einsatz von C'. Die Sprache C wäre dabei nur das Vehikel, um die Organisation von umfangreichen und langlebigen Projekten an konkreten Beispielen zu demonstrieren. Der Leser würde von Anfang an Programme bauen, die sich auch zu umfangreichen Projekten auswachsen können.

Ich habe keine finanziellen Interessen und würde einen Autor gerne mit allen Informationen zu 'Projekt /c' unterstützen.
Es würde mich freuen, wenn andere von meiner Arbeit profitieren könnten.

Workshop

Ein Buch oder der Blog könnte wie ein Workshop aufgebaut werden. Der Teilnehmer würde beim praktischen Erstellen von Programmen das Betriebssystem Linux, die Sprache C , allgemeine Programmierkonzepte und Projektorganisation kennenlernen.

Der Vorteil: Die dabei erstellten Werkstücke können sofort als Basis für eigene Projekte verwendet werden.

Für den Anfang wäre es günstig, nur die einfachen Konzepte aus 'Projekt c/' zu behandeln. Der Autor könnte frei entscheiden, wie tief der Workshop in die Materie eindringt.

Rückfrage

Bei Interesse bitte ich um Rückfrage.

Mein persönliches 'Projekt c/' findet man unter

Webseite	:	http://www.schmuckhexen.at/
Websuche	:	schardinger linux c
eMail	:	v.schardinger@gmx.net

GNU General Public License

```
/*

    Copyright 2022-2025 Günther Schardinger <v.schardinger@gmx.net>

    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/>.
    ____________________________________________________________________________

    Dieses Programm ist Freie Software: Sie können es unter den Bedingungen
    der GNU General Public License, wie von der Free Software Foundation,
    Version 3 der Lizenz oder (nach Ihrer Wahl) jeder neueren
    veröffentlichten Version, weiter verteilen und/oder modifizieren.

    Dieses Programm wird in der Hoffnung bereitgestellt, dass es nützlich sein wird,
    jedoch OHNE JEDE GEWÄHR,; sogar ohne die implizite
    Gewähr der MARKTFÄHIGKEIT oder EIGNUNG FÜR EINEN BESTIMMTEN ZWECK.
    Siehe die GNU General Public License für weitere Einzelheiten.

    Sie sollten eine Kopie der GNU General Public License zusammen mit diesem
    Programm erhalten haben. Wenn nicht, siehe <https://www.gnu.org/licenses/>.

*/
```