Projekt c/ | C - Linux - Arduino - Raspberry

Inhaltsverzeichnis

Einleitung

Das Projekt c/ beschreibt Werkzeuge und Hilfsprogramme für komplexe Hard- und Softwareprojekte für PC, Arduino und Raspberry.

Es geht um Lösungen, die auch im Echtzeitbetrieb laufen und auch nach längerer Zeit noch gut gewartet und erweitert werden können.

Entwicklung und Wartung können auf jedem Linux-System oder über ssh und ssh -X durchgeführt werden.

Die Zielsysteme benötigen nur ein Linux Betriebssystem und eine USB Schnittstelle.

Homepage

Homepage und Downloads: www.schmuckhexen.at/programms

Allgemeine Beschreibungen

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

Dokumentation: C-Programme und Bibliotheken

Die Dokumentationen für C-Programme/Bibliotheken befinden sich in Hilfedateien '1_read.me' und in den C-Headern der Programme/Bibliotheken.

Hilfe zu den fertigen Programmen liefert die Startoption '-h'.

Einstiege:

▷ Programm chelp	Menügesteuerter Zugriff auf alle Dokus
▷ 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

IDE 'geany'

Entwicklungsumgebung für Pi und PC.

Installation

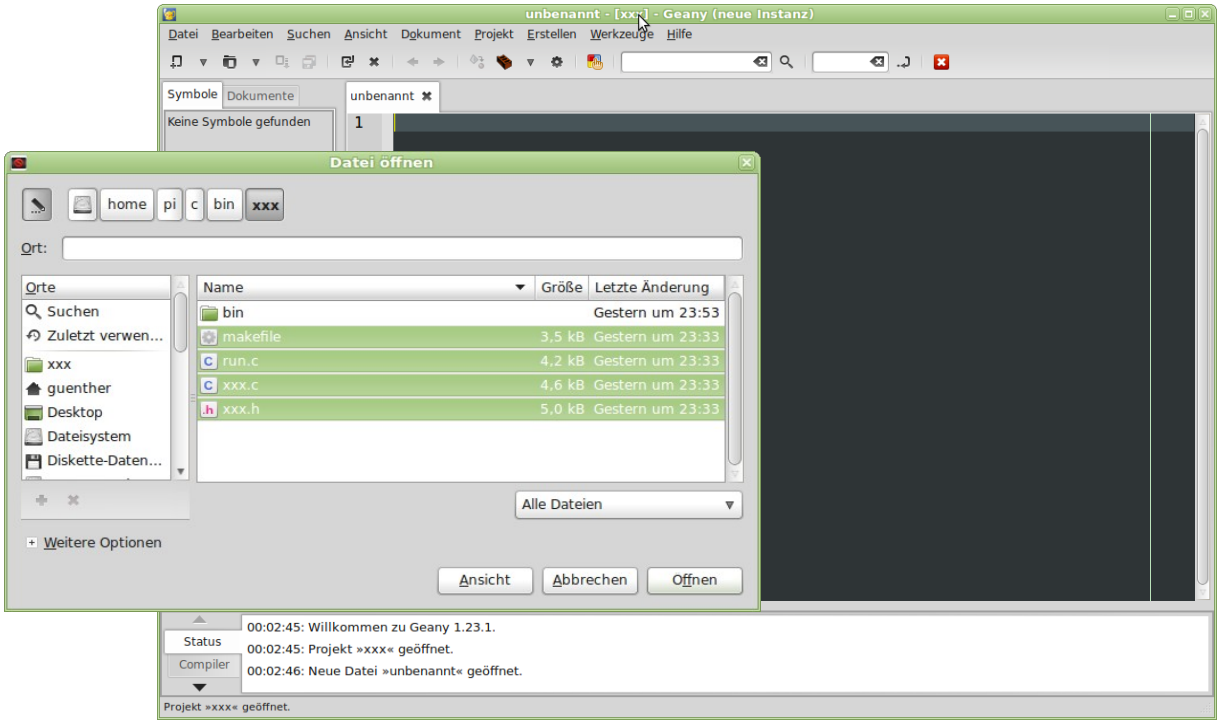
geany ist eine übersichtliche Entwicklungsumgebung für Pi und PC. Sie funktioniert auch sehr gut mit ssh -X am PC.
Die Bedienung ist einfach und der Funktionsumfang ist ausreichend. geany funktioniert auch mit großen Projekten.

Option: Installation von geany

```
sudo apt-get install geany
```

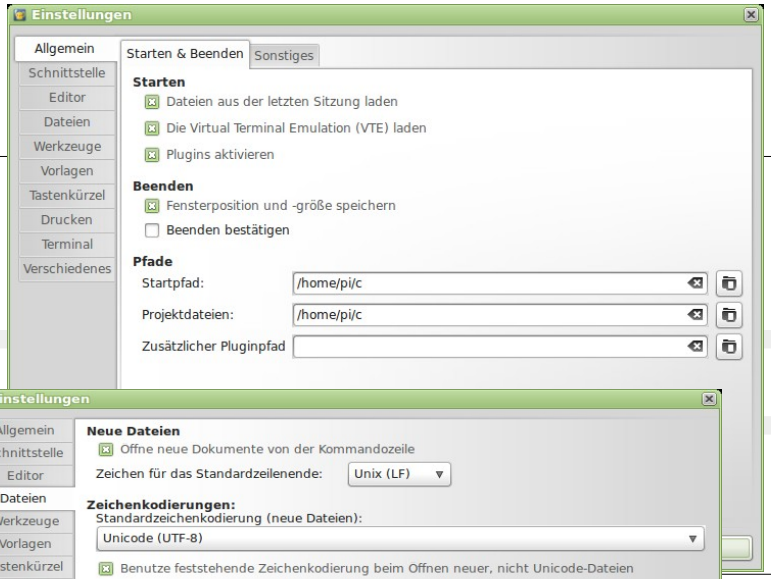
Die Entwicklungsumgebung geany für das Programm xxx kann über chelp oder
den Projektstarter /home/pi/c/bin/xxx.geany gestartet werden.

Beim ersten Start müssen die Projektdateien makefile, xxx.h, xxx.c und xxxx.c mit Datei/Öffnen angezeigt werden.



'geany' konfigurieren

Konfiguration überprüfen:



Einstellungen: Bearbeiten/Einstellungen

Reiter: Allgemein/Starten & Beenden

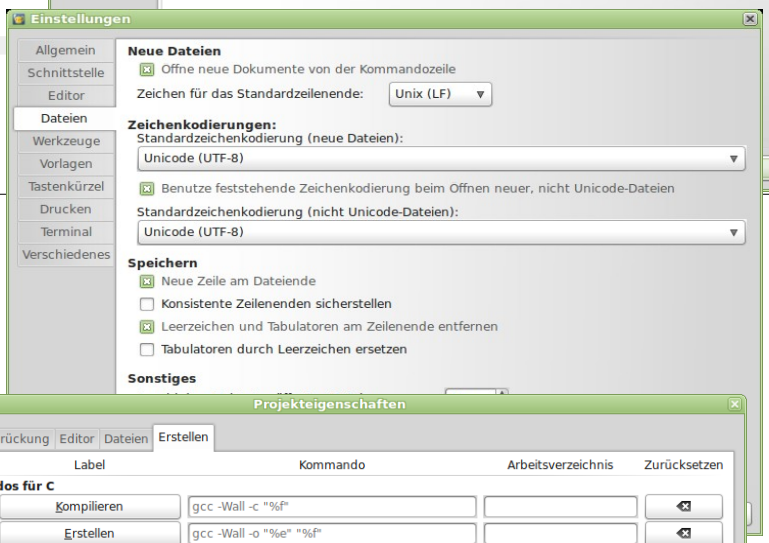
Pfade

Der Startpfad für Pi und PC ist immer gleich:

```
/home/pi/c/....
```

Am Linux-PC wird ein symbolischer Link pi verwendet. Siehe Doku clar_start.pdf :

```
Link /home/pi -> /home/userxy
```



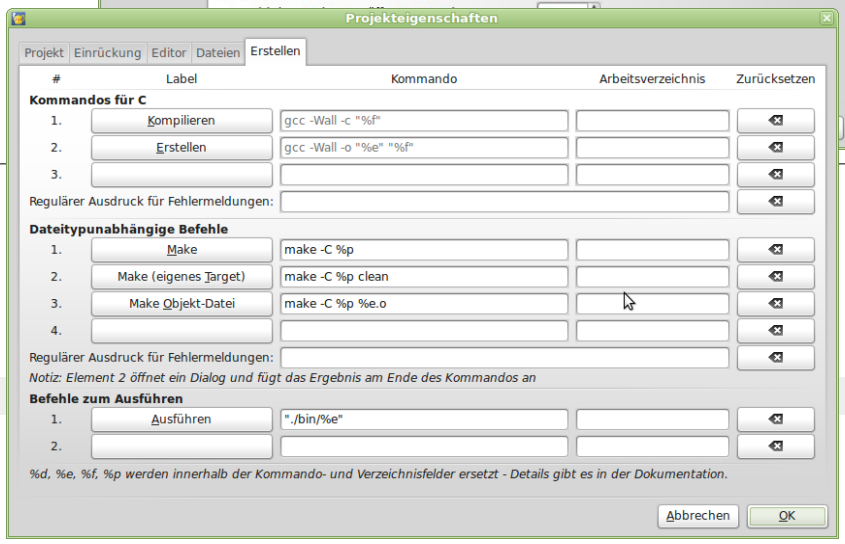
Bearbeiten/Einstellungen: Reiter: Dateien

Zeichenkodierungen

Standardzeichencodierungen: UTF-8
Benutze feststehende Zeichencodierung: ja
Standardzeichenkodierung: UTF-8

Speichern

Leerzeichen am Zeilenende entfernen



Make Einstellungen unter: Erstellen

Kommandos zum Erstellen konfigurieren

Dateitypunabhängige Befehle

Damit wird immer das makefile im passenden Verzeichnis verwendet.

```
make -C %p
make -C %p clean
```

Mit dieser Einstellung funktionieren die make Befehle im Menü oder mit den Tasten:

Erstellen/Make:	oder Taste	Umschalt-F9	für Befehl make
Erstellen/Make (eigenes Target):	oder Taste	Umschalt-Strg-F9	für Befehl make clean
Erstellen/Ausführen:	oder Taste	F5	Programm ausführen

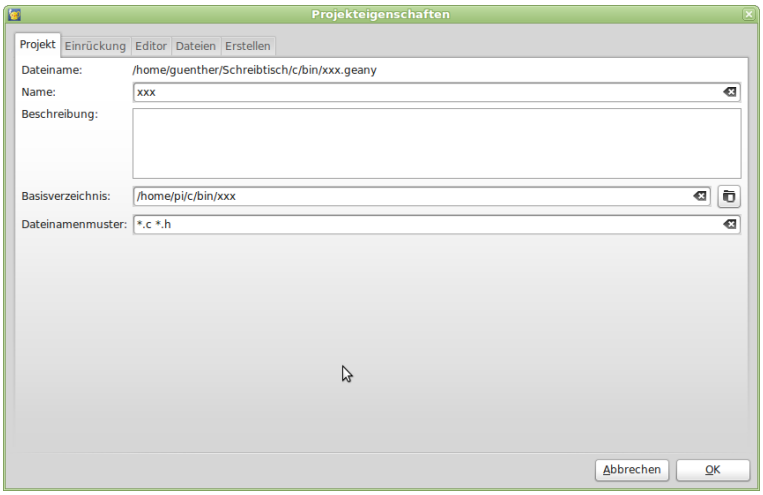
Danach Projekt/Eigenschaften aufrufen.

Basisverzeichnis:

/home/pi/c/pi/bin/xxx

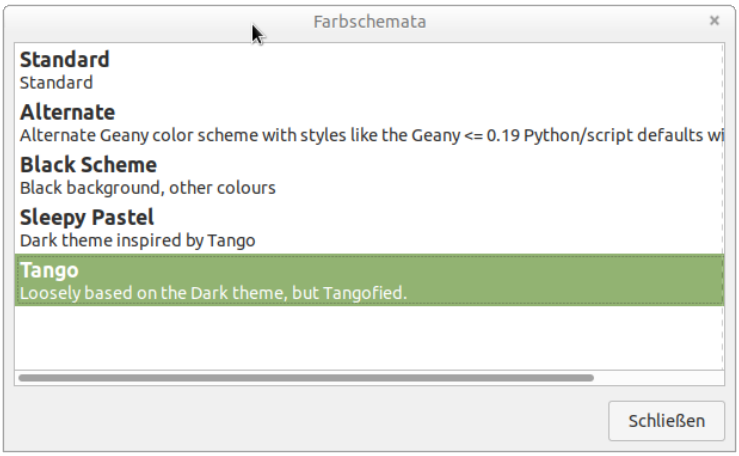
Achtung: Startpfad ist immer /home/pi/ !

Das Geany-Basisverzeichnis sollte nach Doku relative Pfade akzeptieren.
Funktioniert bei mir nicht!



Farbschemata:

Gewüschte Farbschemata z.B. 'Tango' nach /home/guenther/.config/geany/colorschemes kopieren



Vorlagen:

Für Befehl Datei/Neu (aus Vorlage)

Die eigenen Vorlagen können in Ordner /usr/share/geany/templates/files abgelegt werden:

```
├── file.html
├── file_html5.html
├── file.php
├── main.c          // Hauptprogramm
├── main.h          // Hauptheader
├── modul.h         // Header für Module
├── ...
└── NotUsed         // nicht benötigte Vorlagen
    ├── file.rb
    ├── ...
    .
    └── ...
```

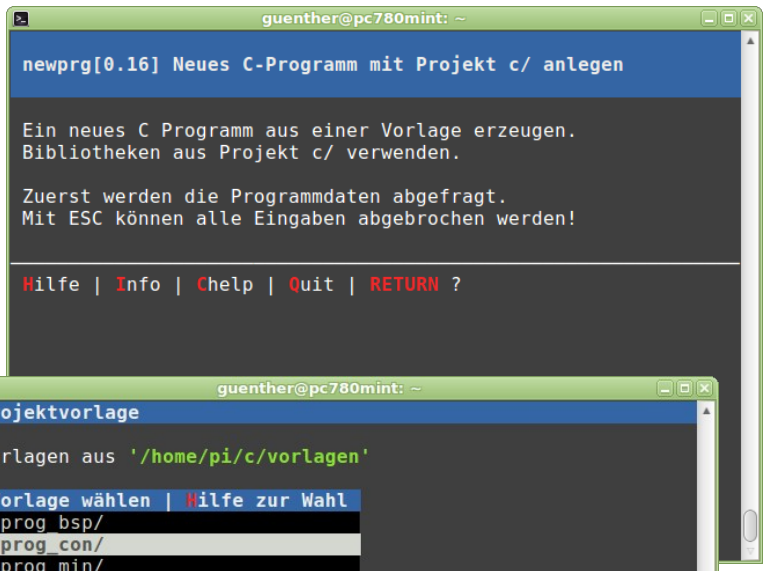
Ein C Programm erstellen

Neue Programme können aus Vorlagen mit Programm newprg erzeugt werden.

Wenn newprg noch nicht compiliert/eingerichtet wurde, kann der Aufruf auch über chelp erfolgen.

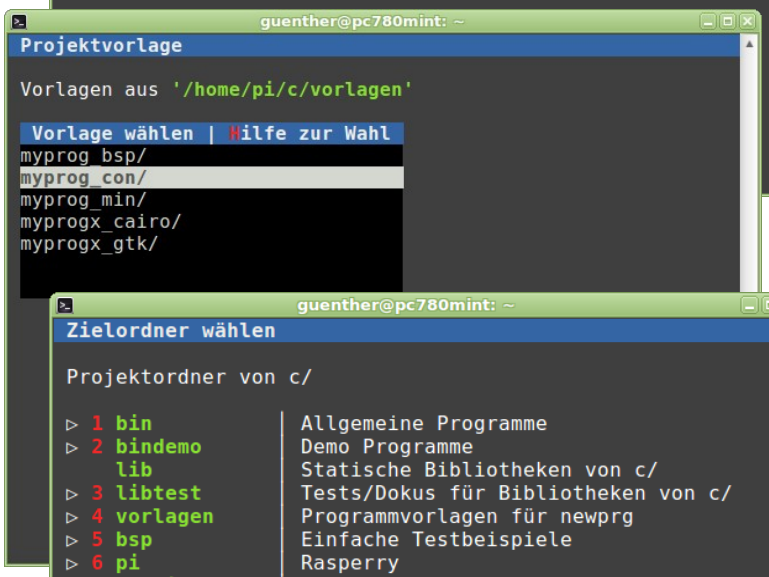
newprg

Mit Programm newprg ein neues C Projekt anlegt.



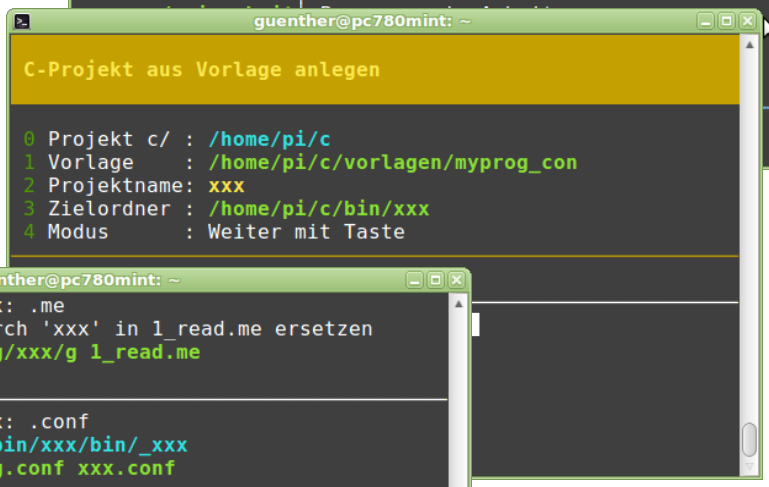
Standardvorlage myprog_con/ auswählen.

Mit Hilfe können Infos zur aktuellen Vorlage angezeigt werden.

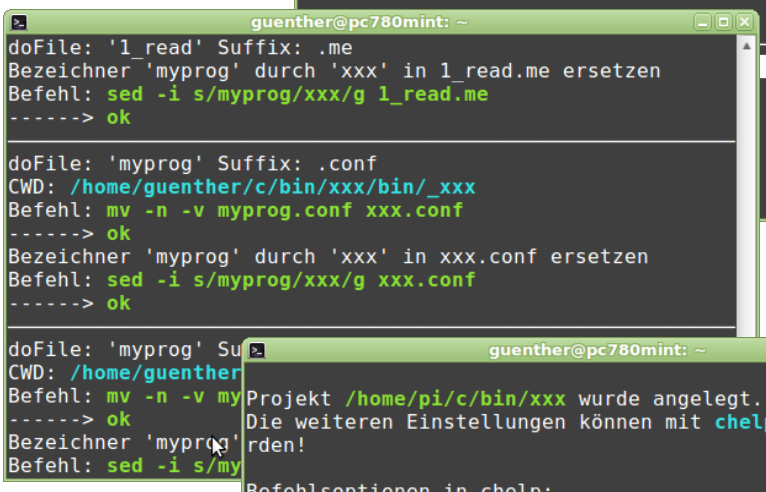


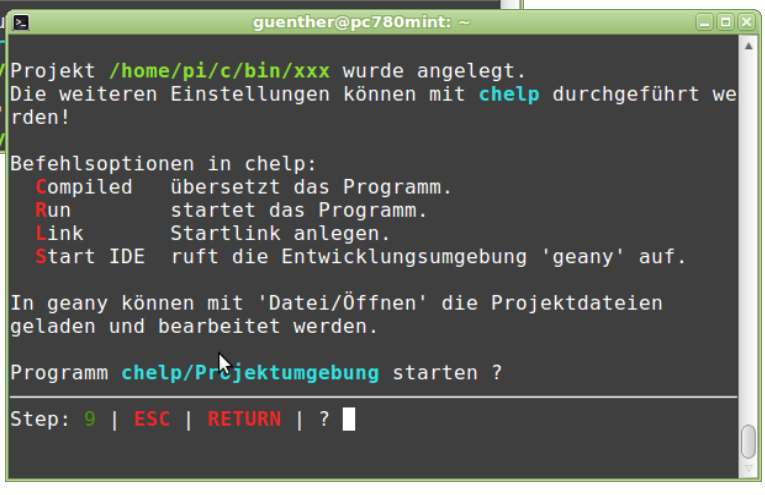
Standard-Zielordner: 1 bin

Zusammenfassung



Neues Projekt anlegen.
Die Durchführung:

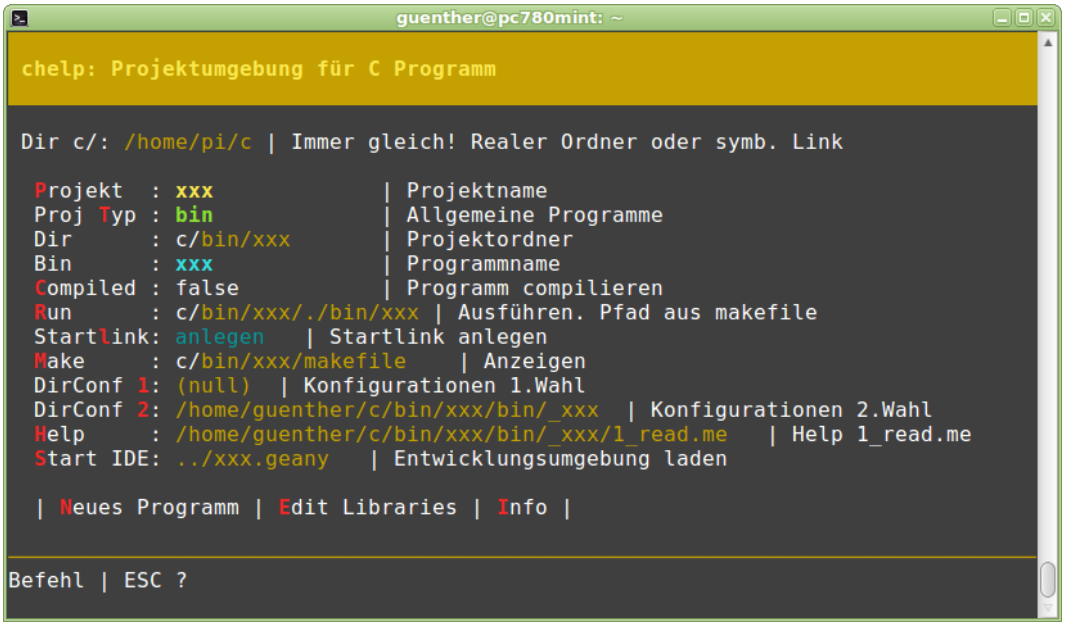




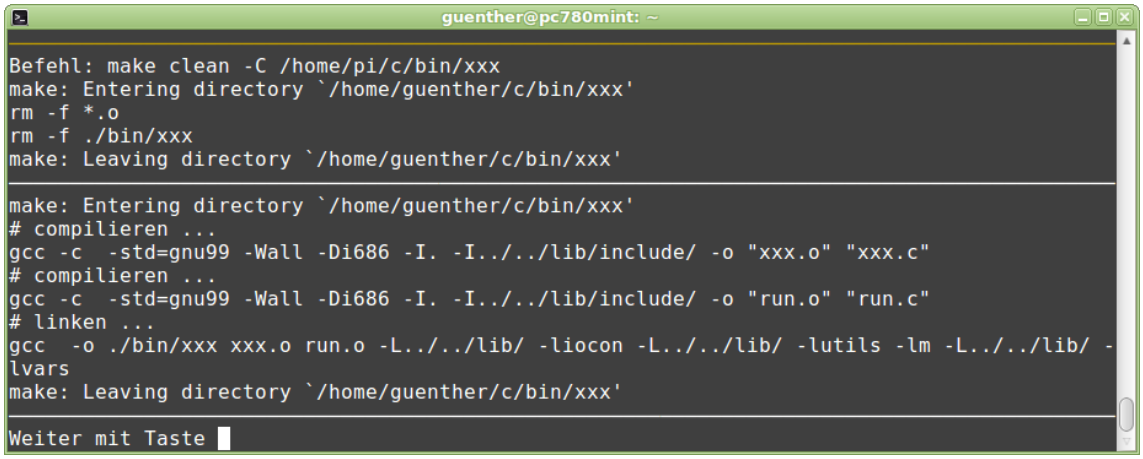
Das neu erzeugte Programm kann mit chelp bearbeitet werden.

Sourcecode

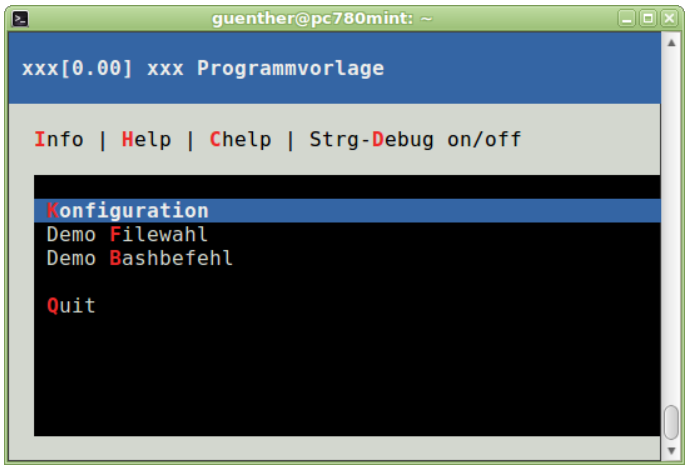
Programm mit chelp einstellen und testen:



Compilieren:
Compiled



Programm ausführen:
Run



Das Programmgerüst kann nun erweitert werden.

Programm bearbeiten:

Zur weiteren Bearbeitung des Programms xxx kann die Entwicklungsumgebung geany verwendet werden.

Befehl: Start IDE

Die Einrichtung/Verwendung von geany wird nachfolgend beschrieben.

Sourcedateien von 'xxx'

test.h Main Header

Includes

```
#ifndef xxx_H
#define xxx_H

#define _GNU_SOURCE    // Linux GNU Erweiterungen nutzen
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
//                     // Libs für Projekt c/ | Header in c/lib/include
#include "utils.h"      // libutils.a | Strings, System- und Hilfsfunktionen
#include "vars.h"       // ibvars.a   | Globale Variablen
#include "script.h"     // libvars.a  | Konfigurations- und Scriptdateien
#include "err.h"        // libutils.a | Fehlerobjekt Err
#include "iocon.h"      // libiocon.a | Input-Output-CONsole oder Terminals.
#include "files.h"      // libutils.a | Dateisystemfunktionen
#include "boxmenu.h"    // libiocon.a | Menüdialog
#include "boxdirwahl.h" // libiocon.a | Ordner-, Device-, oder Dateiwahldialog
```

```
// Testzeile Umlaute ÄÖÜ: Zum Überprüfen der UTF-8 Einstellung des Editors
```

Globale Konstanten, Variablen und Funktionen des Programms

```
// ========================================================================
// Globale Konstanten
//
#define VERSION         "0.00"           // 2020-01-18
#define READMe          "1_read.me"      // Hilfedatei
//
#define CONFIGSuffix0   ".conf"               // Suffix Konfiguration
#define CONFIGSuffix    "*"CONFIGSuffix0      // Suffix Konfiguration
#define CONFIGNameDef   "test"CONFIGSuffix0  // Default Konfiguration
//
#define EDITORDefault   "nano"                // Default Editor
```

```
// ========================================================================
// Globale Variablen
//
uint16_t Debug;
  // 0  : keine Debugausgaben
  // n>0: Level für Debugausgaben
  // Kann durch Aufrufparameter -d oder -D gesetzt werden.
```

Das Var-Objekt verwaltet globale Variablen. Der Zugriff erfolgt über Var-Bezeichner.
Beispiel: VarGetStr(PROGName) liefert den Programmnamen.

```
// ========================================================================
// Var-Objekt für globale Laufzeit-Variablen
//
  // Var-Bezeichner für Variablen aus Var-Objekt
  //
  // Gruppe Programm ------------------------------------------------------
  // Diese Variablen nicht in der Konfigurationsdatei gespeichert
  #define PROGGrp      0              // Gruppe Programmvariablen
  #define PROGSubGrp   0              // Sub-Gruppe Programmvariablen
  #define PROGName     "ProgName"     // Programmname
  #define WORKDir      "WorkDir"      // Arbeitsverzeichnis
  #define CONFIG       "Config"       // Pfad zur aktuellen Konfiguration
  #define EDITOR       "Editor"       // Editor in use
  #define USER         "User"         // Username

  #define DEBUGPrint   "Debug"        // 0,1 oder 2
  //      ...                         //
  //
  // Gruppe Konfiguration -------------------------------------------------
  // Diese Variablen werden aus der Konfigurationsdatei gelesen
  #define CONFGrp      1              // Gruppe für die Konfiguration
  #define CONFSubGrp   0
  #define XEDITOR      "XEditor"      // Var-Bezeichner für Editor für X
  #define CEDITOR      "CEditor"      // Editor für Console

  #define XTERMINAL    "XTerminal"    // Var-Bezeichner für Xterminal
  //      ...
```

Programmfunktionen

```
// ========================================================================
// Deklarationen von Run-Funktionen für Menüaufrufe
//
  void runDemo();           // Ein Befehl
  void runPrintReadMe();    // Hilfedatei 1_read.me anzeigen
  void runPrintConfig();    // Konfigurationsdatei anzeigen
  void runPrintInfos();     // Programminfos
  void runChelp();          // Programminfos
  void runPrintLess();      // Demo PrintLess
```

Globale Hilfsfunktionen

```
// ========================================================================
// Deklarationen globaler Hilfs-Funktionen

  bool callSystemChk(const char *Path);
  // Programm aufrufen mir Fehlercheck

  void printText(const char*TextPfad, const char*Caption, const char *Farbe);
  // Textdatei anzeigen
  ...
```

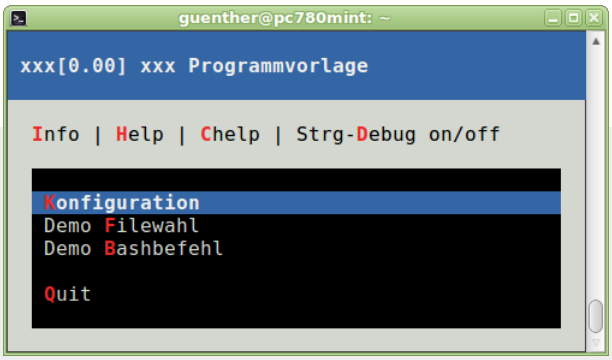
```
  void Exit();                // Exitfunktion. Aufräumen am Programmende
  void ExitSignal(int Signal);  // Abbruch durch Signal. Ruft Exit()

#endif // xxx_H Ende
```

test.c Main Source

```
// Projekt c/ Programm xxx
#include <signal.h>
#include "xxx.h"  // Header xxx
```

```
uint16_t Debug = 0;  // Defaultwert: no Debug
```



Haupmenu

```
// ================================================================
// Definition des Hauptmenus.
// Die vollständige Beschreibung findet man in c/lib/include/boxmenu.h
//
tBoxMenuDef MenuMain=
{ .x=1, .y=1, .h=15, .b=0,         // Menuposition y/x, Höhe/Breite

  .Titel=                          // Titel
  "\n"
  " xxx[" VERSION "] xxx Programmvorlage\n",

  .Info=
  "\n"
  "  "FK"Info | "FK"Help | "FK"Chelp | Strg-"FK"Debug on/off\n",    // Infozeile mit Funktionstasten

  .I=1,                            // Option: Startposition des Cursors

  .FarbeZeile =FarbeGrayBlack,       // Option: Farbe der Menuzeilen
  .FarbeCursor=FarbeWhiteBlueB,      // Option: Farbe des Menucursors


  .Items=(tBoxMenuItem [])         // Menüeinträge
  { {.Typ=Leer },
    {.Txt=" "FK"Konfiguration",   .Typ=Run,  .Ptr=runPrintConfig, .Key='k'},
    {.Txt=" Demo "FK"Filewahl",   .Typ=Run,  .Ptr=runDemo,        .Key='f'},
    {.Txt=" Demo "FK"Bashbefehl", .Typ=Bash, .Ptr="ls -l | less -R -P ' Weiter mit q '", .Key='b'},
    {.Typ=Leer },
    {.Txt=" "FK"PrintLess",       .Typ=Run,  .Ptr=runPrintLess,   .Key='p'},
    {.Typ=Leer },
    {.Txt=" "FK"Quit",            .Typ=Menu  },
    { /*Menuende! Nicht löschen*/ }
  }
};
// ================================================================================
```

Programm initialisieren:

```
void Init()
{ // Variablen und Objekte initialisieren

  clrScr(); hideCursor();

  // Filter für Var-Objekt: Programmgruppe, SubGrp und Filter setzen
  VarSetGrpFlt(PROGGrp,PROGSubGrp,false,false);

  // Programmname und -Dir bestimmen und in Var speichern
  char *Dir, *BaseName;
  getRealDirAndBaseName( getProgPath(), &Dir, &BaseName );
  VarSetStr(PROGName, BaseName);
  VarSetStr(WORKDir,  Dir);

  // in Programmverzeichnis wechseln
  chdir(Dir);

  // Konfigurationspfad bestimmen
  tVar *s=VarSetStr( CONFIG, findConfigPfad( BaseName ,CONFIGNameDef));
  if (ErrChk(Err)) ErrExit("Konfiguration nicht gefunden!\n");

  // Konfiguration lesen und Variablen in Var speichern
  printf(FG);
  if (!readConfig(VarGetpStr(s), CONFGrp, Debug)) ErrExit("Fehler readConfig()\n");

  VarSetGrpFlt(CONFGrp,CONFSubGrp,false,false);  // Alle Vars sichtbar

  atexit(Exit);                // exit() ruft Exit()
  signal(SIGINT,  ExitSignal);  // Option: Aufräumen nach Strg-C
  signal(SIGTERM, ExitSignal);  // Option: Aufräumen nach kill

  if (Debug>0) WeiterMitTaste();
}
```

Hauptprogramm:

```
int main(int argc, char *argv[])
{
  // Kommandozeilenparameter bestimmen
  uint16_t i=1;
  while (i<argc)
  { if (strstr(argv[i], "-h"))
    { printf
      ( "Aufruf: %s [OPTIONS]\n"
        "OPTIONS:\n"
         "  -h     Help\n"
         "  -d, -D Debug\n"
        ,argv[0]
      );
      exit(EXIT_SUCCESS);
    }
    if (0==strcmp(argv[i], "-d")) Debug=1; // Debugmodus
    if (0==strcmp(argv[i], "-D")) Debug=2;
    i++;
  }

  Init();  // Programm initialisieren

  while (true)                                  // Main Loop
  { uint16_t Taste = runMenu(&MenuMain, Debug); // Hauptmenu starten

    switch (low(Taste))            // globale Funktionstasten auswerten
    {
      case 'i': runPrintInfos();   break;
      case 'h': runPrintReadMe();  break;
      case 'c': runChelp();        break;

      case taste_ctl_d:  if (Debug>0) Debug=0; else Debug=1; break;

      case taste_return:  exit(EXIT_SUCCESS);  // Programmende Exit ausführen
    }
  }
} // ================================================================================
```

Exit() wird beim Programmende aufgerufen:

```
void Exit()
{ printLine(0);

  // Heap freigeben: Delete Objekte ...................................
  VarDelete();                 // Delete Var-Objekt

  printf("Exit xxx ...\n");
  printHeapInfo();             // Heapkontrolle
  normal();                    // Textausgabe normal
} // ================================================================================


// ================================================================================
void ExitSignal(int Signal)   // Exit für Signale verwenden
{ Exit();
} // ================================================================================
```

run.c

Beispiel: Konfigurationsfile mit dem Dialog BoxDirWahl bestimmen

```
void runDemo()  // Demo
{ clrScr();
  const char*Path=getFilePathDlg( VarGetStr(CONFIG), "*.bak,*.conf", BoxWahlFile, "Demo Filewahl");

  clrScr();
  printBlock("Demo Filewahl\n", NULL);
  printf
  ("\n" " Filename: %s\n" "\n"
   ,(Path) ? Path : "Filewahl abgebrochen"
  );

  printLine(0);
  WeiterMitTaste();
} // --------------------------------------------------------------------------
```

Hilfsfunktion für den Dialog BoxDirWahl.

```
const char*getStartOrdner()
{ // Ordnerdialog. Rückgabe: Home-Ordner
 return "~";
} // --------------------------------------------------------------------------
```

Aufruf für Dialog BoxDirWahl.
Alle Rückgabestrings sind temporäre Strings. Die temporären Strings werden in einem Ringspeicher angelegt. Der Ringspeicher kann gleichzeitig MaxTmpStr (z.B. 100) verschiedene Strings aufnehmen. Danach wird immer der älteste Strings/Pointer freigegeben. Der String/Pointer bleibt bestehen. Siehe utils.h

```
const char *getFilePathDlg(const char*Start, const char *Suffix, tBoxWahlRet RetTyp, const char*Caption)
{
  if (!Suffix) DirFilterDelete();                    // alle Wildcards löschen
  else         DirFilterSetWildcardsStr(Suffix, ','); // Wildcards neu setzen

  return runBoxDirWahl
  (  1,                   // erste Dialog-Zeile
     1,                   // erste Dialog-Spalte
    -1,                   // Höhe,   <1 vom Bildschirmende
     0,                   // Breite, <1 vom Zeilenende
    Start,                // NULL oder Startordner, Startpath
    DirFilterFiles,       // NULL oder Filter: Sichtbare Ordner und Dateien
    RetTyp,               // Rückgabetyp: Files, Ordner. Siehe tBoxWahlRet
    FCap4,                // NULL Farbe Titel
    Caption,              // NULL oder Titel-Zeilen mit /n getrennt
    getStartOrdner        // NULL oder externer Ordnerdialog
  );
} // --------------------------------------------------------------------------
```

Hilfe-Text anzeigen. Der Dateipfad wird mit tmpStrF(...) temporär angelegt. tmpStrF funktioniert wie prinff(...);

```
void runPrintReadMe()
{ printText(tmpStrF("%s/%s", getDirName( VarGetStr(CONFIG)), READMe),"Help:", FCap1);
} // --------------------------------------------------------------------------
```

Konfiguration anzeigen. VarGetStr(CONFIG) liefert den String-Wert zum Bezeichner CONFIG.

```
void runPrintConfig()
{ printText( VarGetStr(CONFIG), "Konfiguration:", FCap1);
} // --------------------------------------------------------------------------
```

Textfile anzeigen.
Die Anzeige erfolgt mit dem Bash-Befehl cat. Alle Ausgaben zwischen printLessOn(...) und printLessOn(...) werden mit dem pager less angezeigt.

```
void printText(const char*TextPfad, const char*Caption, const char *Farbe)
{ clrScr();

  if (!hasAccess(TextPfad, R_OK))
  { ErrAdd(Err,TextPfad); ErrPrint(Err,"Text nicht gefunden",true); return;
  }
  printLessOn(tmpStrF("%s %s\n", Str0(Caption) ,TextPfad), Farbe, Debug);
  callSystem (tmpStrF("cat %s",TextPfad) );
  printLessOff(NULL, LESSEndText, Farbe, Debug);
} // --------------------------------------------------------------------------
```

Debuginfos anzeigen. Mit VarPrintMitFilterKurz(false) können alle Var-Variablen der gewählten Gruppe angezeigt werden.

```
void runPrintInfos()
{ clrScr();

  printLessOn("\nProgramm-Infos\n", NULL, Debug); // print-Ausgaben speichern

  printf("Version     : %s\n", VERSION );
  printf("System      : %s\n", (isPi()) ? "Raspberry Pi" : "PC" );
  printf("isPi()      : %s\n", BoolToStr(isPi()));
  printf("isXRunning(): %s\n", BoolToStr(isXRunning()));
  printf("Pid         : %i\n", getpid());
  printLn();

  printBlock("Globale Programm-Variablen\n", FCap2);
  VarSetGrpFlt(PROGGrp,PROGSubGrp,true,false);
  VarPrintMitFilterKurz(false);
  printLn();

  printBlock("Globale Config-Variablen\n", FCap2);
  VarSetGrpFlt(CONFGrp,CONFSubGrp,true,false);
  VarPrintMitFilterKurz(false);
  printLn();

  printLessOff(NULL, LESSEndText, NULL, Debug);  // print-Ausgaben mit less anzeigen

  VarSetGrpFlt(PROGGrp,PROGSubGrp,false,false); // Alle Var's sichtbar

} // --------------------------------------------------------------------------
```

makefile

newprg hat das makefile für Programm test angepasst.
Für ein neues Programm müssen die Stellen #|Anpassen| ... geändert werden.
Alle Pfade sind immer relativ vom makefile aus anzugeben.

```
# ============================================================================
# Makefiles Vorlage für Projekte in c. Alle Pfadangaben sind relativ.
# Für neue Projekte die Einträge nach '#|Anpassen|' anpassen!
# Achtung:
# - Zeilenumbruch mit \ aber keine Blanks am Zeilenende!
# - Die Befehle in den Abschnitten Befehle: mit einem Tabulator einrücken!
# ============================================================================
#
# Bedingte Compilierung für PI und PC vorbereiten.
UNAME_M = ${shell uname -m | cut -c1-4}
# Für maschinenabhängige Compilierung wird Compilerflag -D$(UNAME_M) gesetzt.
# System Pi: 'armv' ist definiert.
# System PC: 'armv' ist nicht definiert.
```

```
# 1. Variablen festlegen ====================================================
#
#|Anpassen| Ordner und Programmname ||||||||||||||||||||||||||||||||||||||||||
ZIEL_DIR  = ./bin/
ZIEL      = test

#|Anpassen| Relativer Pfad zu den statischen Libraries ||||||||||||||||||||||
LIBS_DIR = ../../lib/

# Relativer Pfad zu den PC/PI Headerdateien --------------------------------
HEADER_DIR = $(LIBS_DIR)include/

# Namen der statischen Libraries -------------------------------------------
LIB_UTILS = utils
LIB_IOCON = iocon
LIB_VARS  = vars
```

In die Zeile OBJS werden alle Sourcedateien *.c mit dem Suffix *.o eingetragen:
Beispiel: Die Dateien mysource.h und mysource.c werden als mysource.o eingetragen.

```
#|Anpassen| Quelldateien ||||||||||||||||||||||||||||||||||||||||||||||||||||
# INCS    = Optional, weitere Headerdateien *.h
# OBJS    = Ziele *.o . Hier die Namen der Quelldateien *.c eintragen.
INCS      =
OBJS      = $(ZIEL).o run.o mysource.o
# ============================================================================
```

```
# 2. Compiler Einstellungen =================================================
# Einstellungen für den Compiler. Normalerweise keine Änderung notwendig.
# Fehler erscheinen unter '# compilieren ...'
#
CC        = gcc -c
# Compiler: gcc -c oder g++ -c nur compilieren

CFLAGS    = -std=gnu99 -Wall -D$(UNAME_M)
# -Dxx bedingte Compilierung mit define 'xx'
# -std=c99 Standard oder -std=gnu99

INC_DIRS = -I.\
 -I$(HEADER_DIR)
# -Ixxx  xxx ist Pfad zu Headerdateien *.h
# -I.    aktuelles Verzeichnis

DEBUG     =
# -g für Debugger
# ============================================================================
```

```
# 3. Linker Einstellungen ===================================================
# Normalerweise keine Änderung notwendig.
# Fehler erscheinen beim Compilieren unter '# linken ...'
#
LD        = gcc
# Linker:  gcc oder g++ ohne -c
LDFLAGS   =
# Linkerflags

# Pfade zum Linken der Libraries. Achtung: Reihenfolge wichtig!
# Standardpfade sind vordefiniert. Eigene Library brauchen zwei
# Angaben: Ordner xxx -Lxxx und Name yyy ohne lib -lyyy
LIBS  = \
  -L$(LIBS_DIR) -l$(LIB_IOCON)\
  -L$(LIBS_DIR) -l$(LIB_UTILS) -lm\
  -L$(LIBS_DIR) -l$(LIB_VARS)
# ============================================================================
```

```
# Befehle: Ziel duch Linken erstellen =======================================
# Keine Änderung notwendig.
# Abhängig von : Objektdateien
$(ZIEL): $(OBJS)
          # linken ...
          $(LD) $(LDFLAGS) -o $(ZIEL_DIR)$@ $^ $(LIBS)
```

```
# Befehle: Objekt-Dateien compilieren =======================================
# Keine Änderung notwendig.
# Abhängig von : Source- und Headerdateien
%.o : %.c $(INCS)
          # compilieren ...
          $(CC) $(DEBUG) $(CFLAGS) $(INC_DIRS) -o "$@" "$<"
```

```
# Befehle: "make clean" =====================================================
# für Ziel clean: eine Datei 'clean' nicht verwenden!
.PHONY : clean

# Aufruf von "make clean" löscht alle Objektdateien
clean :
          rm -f *.o
          rm -f $(ZIEL_DIR)$(ZIEL)
```

makefile für c/ Programme

Das Standard makefile benutzt bedingte Compilierung. Für die bedingte Compilierung werden 4 Buchstaben des 'machine hardware name' verwendet.

Abfrage am Pi liefert:

```
uname -m | cut -c1-4
armv
```

Abfrage im makefile:

```
UNAME_M = ${shell uname -m | cut -c1-4}
```

UNAME_M hat also am Pi den Wert armv. Mit dem gcc Compilerflag -D kann das Define UNAME_M gesetzt werden:

```
CFLAGS    = -Wall -std=gnu99 -D$(UNAME_M)
```

Anwendung der bedingten Compilierung in Programmen:

Sourcedatei *.c

```
  printf("Bezeichner ");
  #ifdef armv
    printf("'armv6l' definiert       -> Raspberry Pi\n");
  #else
    printf("'armv6l' nicht definiert -> Linux PC\n");
  #endif
```

Anwendung der bedingten Compilierung im makefile zum Linken der passenden Libraries:

makefile

```
# Library: libwiringPi.a  ------------------------------------------------------
# Für Pi und Simulation am PC
LIB_WIRING         = wiringPi

ifeq ($(UNAME_M), armv)
 # System PI: armv6l, Original Lib verwenden
 LIB_WIRING_DIR = /usr/local/lib/
 LIB_WIRING_INC = /usr/local/include/
else
 # System PC: Simulations Lib verwenden
 LIB_WIRING_DIR = ../../libs/wiringpi_sim/
 LIB_WIRING_INC = ../../libs/wiringpi_sim/
endif
```

...

```
# 2. Compiler Einstellungen ========================================================
...

INC_DIRS = -I.\
 -I$(HEADER_DIR)\
 -I$(LIB_WIRING_INC)
# -Ixxx  für Include Pfade der Header
```

...

```
# 3. Linker Einstellungen ==========================================================
...

# Pfade zum Linken der Libraries. Achtung: Reihenfolge wichtig!
# Standardpfade sind vordefiniert. Eigene Library brauchen zwei
# Angaben: Ordner xxx -Lxxx und Name yyy ohne lib -lyyy
LIBS  = \
  -L$(LIB_WIRING_DIR) -l$(LIB_WIRING)\
  -L$(LIBS_DIR) -l$(LIB_IOCON)\
  -L$(LIBS_DIR) -l$(LIB_UTILS) -lm\
  -L$(LIBS_DIR) -l$(LIB_VARS)
# ==================================================================================
```

makefile für Projektordner c/

```
# ========================================================================
# makefile für Projektordner c/...
# Programme und Bibliotheken auf PC und Pi compilieren
#
# Make Aufrufe siehe info:
#
# Achtung: Einrückungen mit einem Tabulator! Keine Blanks am Zeilenende!
# ========================================================================
#
# Bedingte Compilierung für PI und PC:
UNAME_M = ${shell uname -m | cut -c1-4}
# Für maschinenabhängige Compilierung wird Compilerflag -D$(UNAME_M) gesetzt.
# System Pi: define armv ist definiert.
# System PC: armv ist nicht definiert.
# ========================================================================
#
info:
	# ======================================================
	# Make Aufrufe für Projekt /c auf PC und Pi
	# 2020-01-12
	# ------------------------------------------------------
	# Im Projektordner c/
	#
	# make            // Diese Information anzeigen
	# make clean      // Alle Programme und Bibliotheken löschen
	#
	# make allbin     // Bibliotheken und Programme für Pi oder PC
	# make libs       // Nur Bibliotheken erzeugen
	#
	# make chelp      // Hilfe für Projekt c/ compilieren
	# make infosys    // Programm für Systemeinstellungen
	# make pshow      // Bilder und Videos auf der Console anzeigen mit fbp       #
	# make saveit     // Dateien und Ordner auf PC mit Pi synchronisieren
	#
	# make kbdctl     // Songverwaltung für Yamaha PSR-S975
	# make mtrainer   // ALSA-MIDI Test und Gehörtraining
	# make mid2midraw // ALSA-MIDI Gehörtraining
	#
	# make all        // Alles: Libs, Bin, Tests, Demo
	# make Pi         // Pi Programme mit GPIO aus c/pi/
	#
	# ------------------------------------------------------
	# Einzelne Programme compilieren
	# In allen Unterordnern von c/ gibt es makefiles:
	#
	# make            // Program(e) erzeugen
	# make clean      // Program(e) löschen
	# ======================================================


allbin:
	make -C ./lib
	#---------------------------------
	make -C ./bin
	#---------------------------------

all:
	make -C ./lib
	#---------------------------------
	make -C ./arduino
	#---------------------------------
	make -C ./bindemo
	#---------------------------------
	make -C ./bsp
	#---------------------------------
	make -C ./libtest
	#---------------------------------
	make -C ./bin
	#---------------------------------
#	make -C ./vorlagen

Pi:
	make -C ./lib
	#---------------------------------
	make -C ./pi
	#---------------------------------

	#=======================================
	# make all ->Ok
	#=======================================
	# Die Pi Programme aus dem Ordner c/pi/
	# verwenden GPIO (WiringPi).
	# Mit 'make pi' compilieren !
	#=======================================

libs:
	make -C ./lib
	#---------------------------------

chelp:
	make -C ./bin/chelp
	#---------------------------------

infosys:
	make -C ./bin/infosys
	#---------------------------------
```

```
pshow:
          make -C ./bin/fbp
          make -C ./bin/pshow
          #---------------------------------

saveit:
          make -C ./bin/saveit
          #---------------------------------

mtrainer:
          make -C ./bin/mtrainer
          #---------------------------------

mid2midraw:
          make -C ./bin/mid2midraw
          #---------------------------------

kbdctl:
          make -C ./bin/kbdctl
          #---------------------------------


clock:
          # ||||||||||||||||||||||||||||||||||||||||||||||||||||||||
          # |||||||  Programme für sun900 compilieren/installieren
          # |||||||  $(HOME)/bin/clock8583 anlegen
          make -C ./pi/bin/clock8583
ifeq ($(UNAME_M), armv)
          # setuid
          # chmod u+s $(HOME)/bin/clock8583
endif


# 'make sun' für ein Projekt
sun:
          #---------------------------------
          # ||||||| $(HOME)/bin/screenstart anlegen
          make -C ./bin/screenstart
          #---------------------------------
          # ||||||| $(HOME)/bin/sun900 anlegen
          make -C ./arduino/sun_900/sun_900
          #---------------------------------


# Befehl 'make clean' löscht alle Programme und Objektfiles
#
# eine Datei 'clean' nicht als Ziel verwenden
.PHONY : clean
#
clean :
          make -C ./lib clean
          #---------------------------------
          make -C ./arduino clean
          #---------------------------------
          make -C ./bin clean
          #---------------------------------
          make -C ./bindemo clean
          #---------------------------------
          make -C ./bsp clean
          #---------------------------------
          make -C ./libtest clean
          #---------------------------------
          make -C ./pi clean
          #---------------------------------
          make -C ./vorlagen clean
          #---------------------------------

cleanMusic :
          make -C ./bin/kbdctl clean
          make -C ./bin/mtrainer clean
          #---------------------------------
```

Die Projektziele

Systemanforderungen

1. Wenig Installationsaufwand, keine speziellen Systemanforderungen.

   Lösung: Alles befindet sich in einem Ordner. Dieser kann ohne Installation auf die Zielsysteme kopiert werden.
   Am Zielsystem ist nur Linux mit dem C-Compiler erforderlich.
2. Übersichtliche Anzeige in Terminals. Es ist keine grafische Oberfläche notwendig.
   Alles kann über ssh oder ssh -X benutzt werden.
3. Einfache Entwicklung und Wartung von komplexen Projekten.

   Die Stärken der C/C++ Familie:

   - C ist auf jedem Linux verfügbar.
   - Systemnah. Es gibt für alles schon gute Lösungen.
   - Bash-Befehle bieten viele Funktionen an.
   - Make verwaltet sehr inhomogene Sourcen.
   - Compilieren und Testen geht sehr schnell - auch über ssh -X Verbindungen.
   - C ist einfach und die Objekthierarchien bleiben sehr wartungsfreundlich.
   - Copy & Paste von Funktionsblöcken ist sehr einfach und sicher.
   - Mit Vorlagen wird das Erstellen neuer Programme einfach.
   - Übersichtlich geschriebener C-Code ist noch nach Jahren verständlich.

Die Werkzeuge

- Statische C-Bibliotheken: Terminaldialoge, Linux-Systemfunktionen, Raspberry und Arduino.
- Ein universal makefile: Dokumentiert mit bedingter Compilierung.
- Automatische Erstellung von neuen Programmen aus Vorlagen.
- Sichere Kommunikation zwischen PC oder Raspberry und Arduino über USB.
- Fernsteuerung mit ssh oder ssh -X.

Alle C-Lösungen nutzen die vielen Möglichkeiten von GNU-Linux mit Systemfunktionen oder Bash-Befehlen.

- Sichere Stringfunktionen und automatische temporäre Strings.

Durch die Bibliotheksfunktionen werden die Input/Outputfunktionen von C sicher erweitert:

- Alle Inputfunktionen bieten nicht- oder blockierendes IO-Multiplexing
- Low Level Input: peekTaste(), chkTaste(), getTaste().
- High Level Input: Eingabezeile mit der üblichen Cursorsteuerung, Defaultwerte und Fehlerprüfung an.
  readLn(), readDouble(), readInt32(), readUInt32(), readInt16(), readUInt16()
  readHex16(), readHex32()
  readUInt16() und readHex arbeiten auch mit Hex- (0x...) und Binäreingaben (0b...).
- Ausgabe fortlaufend: printf(), printLine(). printBlock() kann farbige Textzeilen in Terminalbreite ausgeben.
- Ausgabe positioniert: printBox() kann farbigen Text in einem Rechteck an beliebigen 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.

Bibliotheksfunktionen bieten sichere und komplexe Systemaufrufe.

- Parametererweiterung und -prüfung. NULL, 0 oder NIL sind gültige Parameter und führen nicht zum Absturz.
- Einheitliche Fehlerbehandlung mit Err-Objekt.
- Komplexe Hilfsfunktionen. Beispiele:

ScanDir() :	Gefilterte Dateilisten von Verzeichnissen. Verwendet scandir().
ScanFile() :	Gefilterte Dateilisten von Verzeichnisbäumen.
Datum/Zeit:	Zugriff und Umrechnungen zwischen den Linux Timeformaten.
openStdLog() :	Die gesamte Terminalausgabe wird in eine Logdatei geschrieben.
getPopenArray(Befehl):	Die Funktion liefert die Ausgabezeilen des Bash-Befehls in einem Array.
printLessOn/Off:	Fortlaufende Terminalausgeban werden mit Pager less angezeigt. Wichtig für Consolen.
usw.	

Für alle Bibliotheksfunktionen gilt:

- Sie akzeptieren immer alle möglichen Funktionsparameter.
- Die String-Rückgaben von Funktionen verwenden temporäre Strings.
- Fehler landen im Error-Objekt Err.
- Objektorientierte Implementierung.
- Jede Funktion kann unabhängig verwendet werden.

GNU General Public License

```
/*
 *
 * Copyright 2020 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 2 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, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
*/
```