Objekte 1 | Basisobjekte | C - Linux - Arduino - Raspberry

Inhaltsverzeichnis

Homepage

Homepage und Downloads: www.projektc.at

Allgemeine Beschreibungen

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

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

Dokumentation: C-Programme und Bibliotheken

- ▷ Die ersten Infos zu den C-Programmen oder Bibliotheken findet man in der Hilfedatei '1_read.me' im jeweiligen Programmordner.
- ▷ Für aufwendige Programme gibt es Beschreibungen im Format *.odt oder *.pdf im Ordner name/bin/_name/
- ▷ Die Dokumentation des Programmcodes befindet sich in den C-Headern der Programme oder Bibliotheken.
- ▷ Hilfe zu den fertigen Programmen liefert immer 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

Library Objekte und Funktionen

Das Projekt c/ besteht aus einer Sammlung von Objekten mit Lösungen für wiederkehrende Programmierprobleme.

Alle Objekte nutzen ausgiebig die von Linux bereitgestellten Bash- und C-Funktionen.

Sie stellen einheitliche und robuste Schnittstellen zur Verfügung und vertragen alle möglichen Aufrufparameter. Insbesondere auch den NULL Pointer.

Diese Anleitung gibt eine kurze Übersicht über einige Library-Objekte und -Funktionen ohne print Ausgaben.
Objekte mit Input/Output Funktionen findet man im 2. Teil clar_objekte2.pdf.

Folgende Statistik vom 2024-09-05 zeigt die tatsächliche Verwendung einiger wichtiger Objekte und Funktionen in c/.

Objekte:

Array	184 mal	ArrayNew()
tmpStr	1835 mal	Temporäre Pointer
Err	1292 mal.	ErrPrint() 295, ErrChkPrint() 531, ErrAdd() 466
Var	1789 mal	VarSetStr() 401, VarGetStr() 927, VarSetInt() 186 , VarGetInt() 275

Systemaufrufe:

callSystem()	397 mal
getPopenArray()	75 mal
callExitAndExec()	26 mal

Stringfunktionen

stripFirst()	186 mal
stripItemI()	119 mal

IO-Funktionen

getTaste()	352 mal
runBoxDirWahl()	54 mal
restPos()	209 mal
printLine()	1448 mal
printBlock()	1127 mal

Sonstige Funktionen

getDate()	145 mal
isPi()	116 mal
freePtr()	2212 mal

Funktionen in Libraries 758

Objekte verwenden

Die Verwendung der Objekte wird am Beispiel des Array-Objekts erklärt.
Dieses wichtige Objekt ist selbst Bestandteil vieler anderer Objekte!

Die formale Beschreibung der Objekte ist immer gleich:

Objektbezeichner	Array	Name des Objekts
Datentyp	tArray	Alle Datentypen werden mit t gekennzeichnet
Objekt anlegen	tArray *a = ArrayNew(...)	Objekt am Heap anlegen. Objekt-Pointer a für den Zugriff.
Objekt Funktionen	ArrayPrint(a)	Objektname: Array Funktionsname: Print , Objekt-Pointer: a
	void *ArrayData(a, i)	Funktionsaufruf
Objekt freigeben	a = ArrayDelete(a)	Heapspeicher freigeben. Rückgabe NULL.

Alle Deklarationen und Beschreibungen der Objekte befinden sich in den zugehörigen Headern im Ordner /home/pi/lib/include.
Die zugehörigen Sourcen befinden sich in den Libraryordnern c/lib/...
Alle mit XxxNew-Funktionen erzeugen Objekte am Heap und müssen mit XxxDelete freigegeben werden.

Siehe nachfolgendes Beispiel Objekt Array.

Beispiel: Objekt Array

Objekt Array verwenden. Zum Erzeugen gibt es die Funktion ArrayNew(...).

Die Projekthilfe chelp bietet verschiedene Möglichkeiten, die Funktionsbeschreibung zu finden:

- Stichworte — Option: 'h| Header | array.h | Library'
- 1 Header aus c/lib/include anzeigen — Option: 4 array.h
- 2 Funktionen in c/ Libraries suchen — Option: . Suche
- 3 String in c/lib/include/*.h suchen — Suchstring oder ESC: ArrayNew(

Man findet den zugehörige Objekt-Header:

```
// ========================================================================
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// |
// | Library  : libvars.a | Arraycontainer für Strings und sonstige Objekte
// | Modul    : array.h
// | Objekt   : tArray
// | Test/Doku: testarray
// |
// | tArray ist ein Container zum indizierten Zugriff auf beliebige Datenitems.
// |  - Der Daten-Zugriff kann über uint32_t Indizes oder Pointer erfolgen.
// |  - Freie Anfangsgröße und automatische Erweiterung.
// |  - tArray kann Daten neu anlegen oder bestehende Daten übernehmen.
// |  - Alle Funktionsaufrufe erwarten einen Arraypointer ungleich NULL.
// |
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// ========================================================================
// Alle Interface-Header sind in $(HEADER_DIR)
// ...
```

Im Kopf des Headers findet man die Library libvars, das Library-Modul array und das Test/Dokuprogramm libtest/testarray .

In der Deklaration wird die Funktion beschrieben:

```
tArray *ArrayNew(size_t StartSize, size_t VerzIncr);
  // Neues Array anlegen.
  // StartSize  : Anfangsgröße. Default: MinVerzIncr
  // VerzIncr   : Verzeichnis wird immer um VerzIncr vergrößert.
  //              (VerzIncr<MinVerzIncr): Default MinVerzIncr.
  //
  // Default Einstellungen:
  //    Verwendung als String-Array
  //    ArrayPrint(): Stringliste anzeigen.
  //    DeleteItem(): Items werden mit freePtr() freigeben.
  //
  // Rückgabe   : tArray
```

Array verwenden

1. Objekt einbinden

Der Header array.h wird mit include in die eigenen Header- oder Source-Dateien eingebunden:

```
include "array.h"
```

2. Objekt verwenden

Array ist für alle Datentypen geeignet!
Beispiel: Stringarray a mit Defaulteinstellungen erzeugen:

```
tArray *a = ArrayNew(0,0);
```

3. Stringarray verwenden.

Zwei Strings am Array-Ende Einfügen. ArrayAddStr(a, ...) fügt Kopien der Strings am Heap ein.

```
ArrayAddStr(a,
    "xxx Zeile 1\n"
    "zzz Zeile 2\n"
);
```

Bestehenden String übernehmen. Keinen eigenen Heapspeicher verwenden:

```
char *s=newStr("aaa Zeile 3");  // String s am Heap
ArrayAddPtr(a, s);              // String übernehmen
s="yyy Zeile 4";                // Konstanter String s
ArrayAddPtr(a, s);              // String übernehmen
```

Array anzeigen.

ArrayPrint(a) verwendet die Default Print-Funktion ArrayItemStr(). Mit eigenen Print-Funktionen können beliebige Datentypen angezeigt werden.

```
ArrayPrint(a);
```

Ausgabe:

```
xxx Zeile 1
zzz Zeile 2
aaa Zeile 3
yyy Zeile 4
```

Array sortieren.

ArrayQSort(a,..) verwendet die Default Sortierfunktion ArraySortCompareStr(). Es können eigene Sortierfunktionen verwendet werden.

```
ArrayQSort(a, NULL ); // NULL für Strings. Es wird die Sortierfunktion ArraySortCompareStr verwendet.
ArrayPrint(a);
```

Ausgabe:

```
aaa Zeile 3
xxx Zeile 1
yyy Zeile 4
zzz Zeile 2
```

String suchen und anzeigen.

ArrayFind(a, Findfunktion, String) sucht den String mit Default Findfunktion ArrayCompareStr(). Es können eigene Findfunktionen verwendet werden.
Rückgabe: Der Array Index des Strings oder NIL für nicht gefunden.
Anzeige : Arrayelement mit ArrayPrintItem(a,i) ausgeben.

```
uint32_t i=ArrayFind(a, ArrayFindCompareStr, "zzz Zeile 2");
printf("%u: %s\n",i, ArrayPrintItem(a, i));
```

Ausgabe:

```
3: zzz Zeile 2
```

Zugriff auf einzelne Arrayelemente mit dem Index.

ArrayData(a, i) liefert einen Pointer auf das Daten-Item oder NULL.
Hinweis: printf(...) verweigert manchmal die Ausgabe von NULL. StrN(NULL) gibt sicher "NULL" zurück.

```
printf("%s\n", StrN( ArrayData(a, 3) ));   // gültiger Index
```

Ausgabe:

```
zzz Zeile 2
```

```
printf("%s\n", StrN( ArrayData(a, 20) ));  // ungültiger Index
```

Ausgabe:

```
NULL
```

Alle Arrayelemente durchlaufen und anzeigen.

```
for (uint32_t i=0; i<ArraySize(a); i++) printf("%s\n", ArrayPrintItem(a,i));
```

4. Objekt freigeben.

Funktion ArrayDelete() verwendet die gesetzte Funktion ArrayDeleteItemX(), um den Speicher für jedes Datenelement freizugeben. Danach wird das Array-Objekt selbst freigegeben. Rückgabe NULL.

Für Stringarrays ist für zur Freigabe von Datenelementen die Funktion ArrayDeleteItemFree() die Defaulteinstellung.
Die vordefiniertOption ArrayDeleteItemNop() überspringt die Freigabe von Datenelementen.
Für selbst definierte Datenelemente muss mit ArraySetDelete(a, DeleteItem) eine passende DeleteItem-Funktion gesetzt werden.

```
a=ArrayDelete(a);     // a zur Sicherheit NULL setzen
```

Library: utils.a

Objekte und Funktionen aus den Modulen der Library utils.a.
utils.a: Stellt Funktionen für Strings, Systemabfragen und -aufrufe bereit.
Informationen liefert der Projektbrowser chelp Stichwort 'Lib utils.h:'

Strings | temporäre Pointer

Für temporäre Strings/Pointer gibt es das statische tmpStr-Objekt. Es gibt den verwendeten Heapspeicher automatisch wieder frei.

Das tmpStr-Objekt wird automatisch angelegt. Die Pointer werden fortlaufend in einem Ringspeicher mit maximal MaxTmpStr verschiedenen Elementen abgelegt. Wenn der Ringspeicher voll ist, wird immer der älteste Pointer wieder freigegeben.

MaxTmpStr ist in utils.h definiert: #define MAXTmpStr 300

In Projekt c/ geben alle Stringfunktionen immer temporäre Strings zurück. Stringmanipulationen und Parameterübergaben funktionieren damit normalerweise problemlos! Nur temporäre Strings in Wiederholungen, die außerhalb der Wiederholung definiert wurden, sind unzulässig. Diese Fehler sind rasch durch 'eigenartiges Programmverhalten' zu erkennen.

```
void FehlerDemo()
 {
   const char *s_tmp = tmpStr("Problem");
   const char *s_fix = newStr("Kein Problem");

   while(...)
   {  ...
      ... = s_tmp;       // dieser Zugriff ist nach MAXTmpStr Durchläufen möglicherweise undefiniert!
      ... = s_fix;
   }
   s_fix=freePtr(s_fix); // s_fix freigeben
 }
```

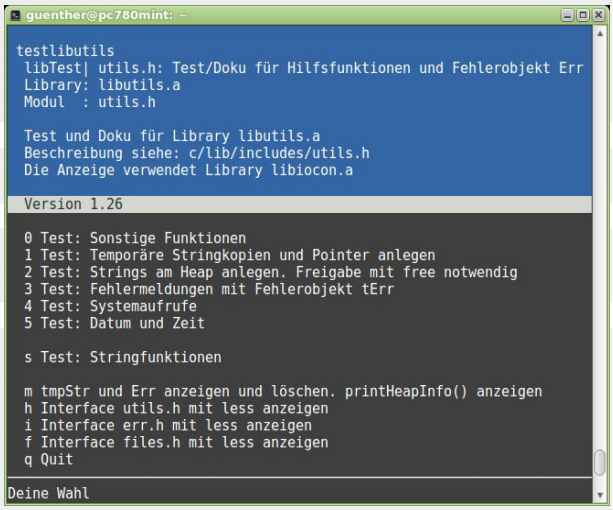
tmpStr-Objekt kann auch verwendet werden, um Pointer/Objekte zu einem späteren Zeitpunkt automatisch freizugeben.

Beschreibungen chelp: Stichworte/ Lib utils.h: Temporäre Strings, Stringfunktionen
Modul: c/lib/include/utils.h
Testprogramme: c/libtest/testlibutils

Temporäre Strings und Pointer:

Testprogramm: c/libtest/testlibutils

```
// ============================================================================
// tmpStr| Temporäre Strings mit automatischer Freigabe  ||||||||||||||||||||
//
 define MaxTmpStr 300
// Die temporären Strings werden in einem Ringspeicher angelegt.
// Der Ringspeicher kann gleichzeitig MaxTmpStr verschiedene Strings
// oder sonstige Pointer aufnehmen. Danach wird immer der älteste
// Strings/Pointer freigegeben. Der String/Pointer bleibt bestehen.
//
// tmpStr's werden immer automatisch freigegeben. Kein free() verwenden!

char *tmpStr(const char *String);
  // if (String) Rückgabe: Pointer auf temporäre Kopie von String.
  // else        Rückgabe: NULL

char *tmpStrN(const char *String, size_t n);
  // if (String) Rückgabe: Pointer auf temporäre Kopie von String.
  //                       Maximal n Bytes kopieren. Ende '\0' hinzufügen
  // else        Rückgabe: NULL

char *tmpStrF(const char *str, ...);
  // Temporären formatierten String anlegen.
  // Die Formatierung erfolgt wie mit printf().
  //
  // Eingabe:
  //     *str: Formatstring wie für sprintf()
  //     ... : Parameter für den Formatstring
  //           Intern wird vasprintf( sptr ,str, ...) verwendet.
  //
  // Rückgabe : Pointer auf formatierten String oder NULL im Fehlerfall
  // Beispiel : int i=25;  char *p=tmpStrF("Integerwert=%i", i);
  // Fehler   : tmpStrF("... %s ...") ohne Argumente!

char *tmpStrPtr(size_t size);
  // Temporären Heapspeicher der Größe size+1, gefüllt mit 0, anlegen.

void *tmpAddPtr(void *Ptr);
  // Heappointer für die spätere Freigabe übernehmen.
  // Rückgabe: Ptr
```

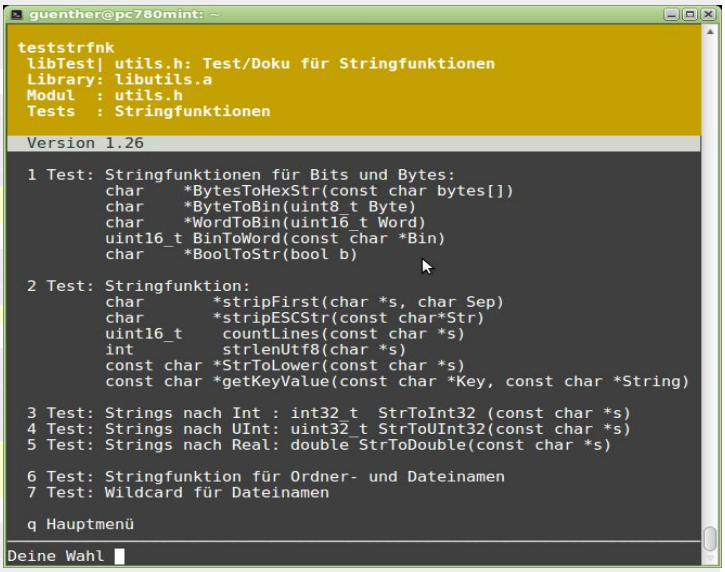


Strings am Heap

Funktionen für dauerhafte Strings am Heap:

Testprogramm: c/libtest/testlibutils

```
char *newStr( const char *String);
  // Stringkopie am Heap anlegen. Verwendet strdup().
  // Freigabe mit freePtr() notwendig.
  // String==NULL: Rückgabe NULL

char *newStrF( const char *str, ...);
  // Neuen String mit Formatstring am Heap anlegen.
  // Freigabe mit freePtr() notwendig.
  // Fehler: newStrF("... %s ...") ohne Argumente!
  //         newStr() verwenden!
  // Beispiel: int i = 256; s = setStr("i=%i", i);
  //           --> s ist "i=256"

char *setStr(char *s, const char *NewString);
  // Kopie von NewString am Heap anlegen und s freigeben.
  // Freigabe mit freePtr() notwendig.
  // Beispiel: s = setStr(s ,"Neu") --> s ist "Neu"

char *setStrF(char *s, const char *Format, ... );
  // Neuen String mit Formatstring erzeugen und s freigeben.
  // Freigabe mit freePtr() notwendig.
  // Fehler: setStrF("... %s ...") ohne Argumente!
  //         setStr() verwenden!
  // Beispiel: s = newStr("Ergebnis");
  //           s = setStrF(s, "%s = %i", s, 25);
  //           --> s ist dann "Ergebnis = 25"

char *addStr(char *s, const char *s1);
  // Kopie von String s+s1 am Heap erzeugen und s freigeben.
  // Freigabe mit freePtr() notwendig.
  // Beispiel: s = newStr("String 1");
  //           s = addStr(s ,", String 2");
  //           --> s ist "String 1, String 2"
```




```
void *newPtr(size_t Size);
  // Speicher am Heap anlegen mit 0 füllen. Freigabe mit freePtr() notwendig.
  // Size==0: Rückgabe NULL. Verwendet calloc().
  // Programmabbruch mit Fehler bei Speichermangel.

void *freePtr(const void *ptr);
  // Heapspeicher zu ptr freigeben. Ersetzt free()
  // Kann mit auch Heap, Text und Stackpointer verwendet werden!
  // Siehe Programm infosys, Befehl '5 Memory'
  // ptr==NULL             Rückgabe NULL
  // ptr am Heap           free(ptr) und Rückgabe NULL
  // ptr im text segment   Rückgabe ptr
  // ptr am Stack          Rückgabe ptr | Achtung: Funktioniert nicht unter valgrid
  // Verwendet isHeapPtr().


typedef char *tStrLst[]; // Stringliste
```

Err | Fehlerobjekt

Das Fehlerobjekt Err deckt mit wenigen Funktionen eine reichhaltige Fehlerbehandlungen ab. Alle Fehler sollen entdeckt werden, aber nur an passender Stelle gemeldet werden. Die Fehlermeldung sollte dem Benutzer/Programmierer eine sofortige Problemlösung ermöglichen.

Fehlerobjekt Err wird immer automatisch mit dem ersten Funktionsaufruf angelegt. Im Folgenden soll das Zusammenspiel der Fehlerfunktionen werden.

Fehler sammeln: Fehler aus verschiedenen Funktionen können gesammelt und gespeichert werden.

ErrAdd(Err, "Meldungen")	Systemfehler abfragen und im Klartext an die Fehlerliste anhängen. Danach werden die eignen "Meldungen" angehängt. Das Programm läuft weiter. Fehleranzeige und Behandlung können später erfolgen.

Fehler anzeigen: Mit den folgenden zwei Funktion können Fehler angezeigt oder in einem Logfile gespeichert werden.

ErrPrint(Err, "Meldungen", WarteAufTaste)	Systemfehler abfragen und alle gespeicherten Fehler und "Meldungen" anzeigen. WarteAufTaste == true: Fehleranzeige zeigen und anhalten. Weiter mit WeiterMitTaste().
ErrChkPrint(Err, "Meldungen", WarteAufTaste)	Nur wenn Fehler anliegen, ErrPrint(). aufrufen

Fehleranzeigen steuern:

ErrSetPrint(tErr, On)	On	false, Fehleranzeige aufschieben. Die Printfunktionen zeigen nichts an, sondern führen nur ein ErrAdd aus.

In autonom laufenden Steuerungen ist es nicht günstig, wenn ein Programme endlos auf eine Benutzereingaben wartet. Die Funktion WarteAufTaste kann daher überschrieben werden. In Fehleranzeigen mit less und LogFile wird WarteAufTaste normalerweise automatisch übersprungen. Für Sonderfälle kann das Verhalten der Fehleranzeige aber genau eingestellt werden.

ErrSetWarteAufTaste(Err, WarteSec)	Mit dieser Funktion kann das Verhalten von WarteAufTaste eingestellt werden.		
	WarteSec:	n>=0	Wartezeit nach n Sekunden automatisch beenden.
		-1	Wartezeit ohne Logfile unendlich. Defaultwert.
		-2	Wartezeit immer unendlich auch mit Logfile!.
	Rückgabe:		Letzte WarteSec Einstellung. Damit kann WarteAufTaste in Funktionen temporär geändert werden.
ErrSetPrintMode(Err, ScrOn, LogFile)	Mit dieser Funktion können die Ziele der Fehlerausgabe eingestellt werden.		
	ScrOn:		true, Fehler (auch) am Bildschirm anzeigen
	LogFile:		Bei LogFile!=NULL, werden die Fehler in das LogFile geschrieben. WarteAufTaste wird dann normalerweise übersprungen. Für Sonderfälle kann mit WarteSec=-2 das Standardverhalten WarteAufTaste .

Informative Fehlermeldungen erzeugen:

Damit lassen sich Fehler gut lokalisieren lassen, ist es günstig den Funktionsnamen mit __func__ in die Fehlermeldung zu setzen.
Aus dem Testprogramme für Err: c/libtest/testlibutils

```
▶ Meldungen ablegen:
  ErrAdd(Err,tmpStrF("%s | Meldung 0 ",__func__))
  ErrAdd(Err,"Meldung 1\nMeldung 2")
```

Fehler ablegen mit
ErrAdd(Err, tmpStrF("%s | Meldung 0 ", __func__)
ErrAdd(Err, "Meldung 1\nMeldung 2 ")

```
Obj Err: 0x55b1e0a572a0, default
 PrintOn  : 1   | 1: Print sofort, 0: nur ErrAdd
 ScrOn    : 1   | 1: Err Ausgabe auch Bildschirm
 LogFile  : 0   | 1: Err ins Logfile schreiben
 errno    : 0   | Letzter Systemfehler
 WarteSec : -1  | Wartezeit in Fehleranzeige. -1 endlos, -2 endlos mit Logfile
 Meldungen: 3   | Gespeicherte Err-Meldungen
 0: TestErrObjekt | Meldung 0
 1: Meldung 1
 2: Meldung 2
```

Infos zum Fehlerobjekt mit ErrPrintInfo(Err) anzeigen

```
▶ Fehler und Meldungen anzeigen, löschen und warten:
  ErrPrint(Err,"Meldung 3",true )
TestErrObjekt | Meldung 0
Meldung 1
Meldung 2
Meldung 3
```

Fehler anzeigen mit ErrPrintInfo(Err)

Der vollständige Header zu tErr:

Beschreibungen chelp: Stichworte/ Lib err.h : Err-Objekt
Modul: c/lib/include/err.h
Testprogramme: c/libtest/testlibutils

```
// ========================================================================
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// |
// | Library  : libutils.a | Fehlerobjekt Err für Projekt c/
// | Modul    : err.h
// | Objekt   : tErr
// | Test/Doku: test_libutils, c/Dokus/clar_objekte2.odt
// |
// | Fehlerobjekt:
// | - Meldungen und Systemfehler speichern
// | - Meldungen und Systemfehler anzeigen
// | - Meldungen und Systemfehler checken
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// ========================================================================
```

```
typedef struct tErr tErr;  // private Struktur

extern tErr *Err;          // Default Err-Objekt. Anfangswert NULL.

tErr *ErrNew();
  // Standard-Objekt Err wird bei ersten Funktionsaufruf automatisch erstellt!
  // Option: Ein eigenes Fehlerobjekt neben tErr erstellen.

tErr *ErrDelete(tErr *p);
  // Ein Error Objekt freigeben.

// ============================================================================
// Die Fehlerfunktionen

bool ErrClr(tErr *p);
  // Gespeicherte Meldungen und errno löschen.
  // Objekt bleibt erhalten.
  // Rückgabe: true : Es gab Fehler
  //           false: Es gab keine Fehler


bool ErrChk(tErr *p);
  // Gespeicherte Meldungen prüfen
  // Systemfehler werden mit ErrAdd() gespeichert.
  // Es werden keine Meldungen angezeigt!
  // Rückgabe true: Es gibt Systemfehler oder gespeicherte Meldungen.


void ErrAdd(tErr *p, const char* Lines);
  // Fehler merken
  // if (errno!=0): Systemfehlermeldung hinzufügen und errno=0 setzen.
  // if (Lines)   : Eine oder mehrere Meldungszeilen hinzufügen.
  //                '\n': neue Zeile und Farbe bis zum rechten Rand.
  // Meldungen werden nicht angezeigt!

bool ErrChkAdd(tErr *p, const char*Lines);
  // Fehler prüfen/merken
  // if ( ErrChk() ):  ErrAdd() aufrufen und Rückgabe true;
  // else              Rückgabe false;


void ErrPrint(tErr *p, const char* Lines, bool WarteAufTaste);
  // Fehler anzeigen/löschen
  // Alle Fehlermeldungen und Lines anzeigen und löschen:
  // - gespeicherte Meldungen
  // - Systemfehler
  // - Lines
  // WarteAufTaste true: Ausgabe anhalten und Taste maximal WarteSec warten
  //                     Bei Ausgabe mit lessPrint() nie anhalten!


bool ErrChkPrint(tErr *p, const char *Lines, bool WarteAufTaste);
  // Auf Fehler prüfen und anzeigen/löschen.
  // if ( ErrChk() ):  ErrPrint() aufrufen und Rückgabe true;
  // else              Rückgabe false;
  // Die Lines werden nicht gespeichert.
  // Gespeicherte Meldungen werden mit ErrClr() gelöscht.
  // WarteAufTaste true: Ausgabe anhalten und Taste maximal WarteSec warten
  //                     Bei Ausgabe mit lessPrint() nie anhalten!


int16_t ErrSetWarteAufTaste(tErr *p, int32_t WarteSec);
  // Wartezeit nach einer Fehleranzeige mit den Funktionen ErrPrint(..., true)
  // und ErrChkPrint(..., true) für WeiterMitTasteSec(WarteSec,...) setzen.
  // Bei Fehlerausgaben ins Logfile gibt es für WarteSec>=-1 keine Wartezeit!
  //
  // WarteSec: n>=0  Wartezeit nach n Sekunden automatisch beenden
  //           -1    Wartezeit ohne Logfile unendlich
  //           -2    Wartezeit auch mit Logfile immer unendlich!
  //
  // Rückgabe:  Letzter Wert von WarteSec


void ErrSetPrintMode(tErr *p, bool ScrOn, FILE *Logfile);
  // Ausgabemodus für Bildschirm und/oder Logfile setzen.
  //
  // ScrOn  : true   Err-Ausgabe immer auch am Bildschirm mit Farbe-Esc-Sequenzen.
  // Logfile: !NULL  Err-Ausgabe ins Logfile ohne Farb-Esc-Sequenzen.
  //                 Mit WarteSec>=-1 keine Wartezeit Fehleranzeige
  //           NULL  Err-Ausgabe nur am Bildschirm.
  //


void ErrSetPrint(tErr *p, bool On );
  // Fehlerausgabe ein/auschalten
  // On==true  : ErrPrint() und ErrChkPrint() drucken/speichern
  // On==false : ErrPrint() und ErrChkPrint() führen nur ErrAdd() aus


void ErrExit(const char *Meldung); // Fehleranzeige und Programmabbruch

// ============================================================================
// Debugfunktionen für tErr

void ErrPrintInfo(tErr *p); // Infos zum Fehlerobjekt anzeigen

// Vordefinierte Makros für Debug
// Sie verwenden die globale Variable 'uint16_t Debug' und
// verkürzen den Befehl "if (Debug>0) printf" zu "DP1"
//
#define DP1(...) if (Debug>0) printf(__VA_ARGS__)
#define DP2(...) if (Debug>1) printf(__VA_ARGS__)
#define DP3(...) if (Debug>2) printf(__VA_ARGS__)

// Beispiel:  Befehl:   if (Debug>0) printf(FV"%s Anf=%u\n"FN,__func__ ,Pos);
//            wird zu   DP1(FV"%s Anf=%u\n"FN,__func__ ,Pos);
```

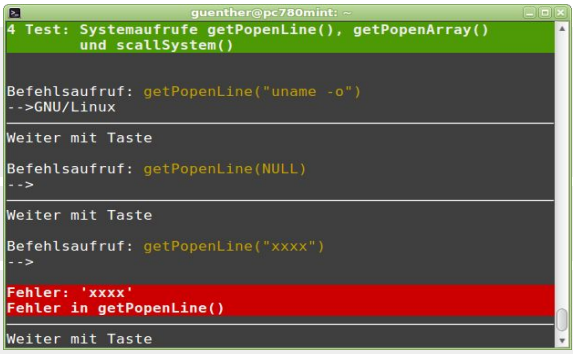
Systemaufrufe

Beschreibungen chelp: Stichworte/ Lib utils.h: Temporäre Strings, Hilfsfunktionen
oder 1 Header | 1 utils.h
Modul: c/lib/include/utils.h
Testprogramme: c/libtest/testlibutils | c/libtest/testsystem

```
//=====================================================================
// Systemaufrufe         |||||||||||||||||||||||||||||||||||||||||||||||
// mit Fehlermeldungen   |||||||||||||||||||||||||||||||||||||||||||||||

const char *getPopenLine(const char *Befehl);
  // Bash-Befehl mit popen() aufrufen.
  // Rückgabe: Erste Antwortzeile ohne '\n'
  // Fehler  : Rückgabe "" und Fehler im Err-Objekt
```

```
uint32_t getPopenArray(const char *Befehl, tArray *a);
  // Bash-Befehl mit popen() aufrufen.
  // Rückgabe: Zeilenanzahl und Array mit Antwortzeilen
  // Fehler  : Fehler im Err-Objekt
```

```
bool callSystem(const char *Befehl);
  // Der Bash-Befehl wird im aktuellen Ordner mit system(Befehl)
  // aufgerufen. Das aktuelle Programm läuft weiter.
  // Achtung : Befehle 'xxx &' melden keine Fehler an das Programm!
  //           Zugriff mit isCommand() vorher prüfen.
  // Rückgabe: false, Fehler oder Return-Status>0 im Err-Objekt
```




```
bool callExitAndExec(tExitFnkt ExitFnkt, char *const Opt[], uint16_t Debug);
  // Das aktuelle Programm vollständig durch ein anderes Programm ersetzen.
  //  - Die Exitfunktion im aktuellen Programm ausführen.
  //  - Das das neue Programm Opt[0] mit Options Opt[1] bis Opt[n] starten.
  //  - Das aktuelle Programm beenden.
  //
  // Eingaben:
  //   ExitFnkt NULL oder Exitfunktion des aktuelen Programms.
  //            ExitFnkt sollte die Konfiguration speichern,
  //            mit fclose() alle Files schließen und den Heap freigeben!
  //            ExitFnkt!=NULL: Programm wird über exit() mit Exit() beendet
  //            ExitFnkt==NULL: Programm wird mit _exit() ohne Exit() beendet
  //
  //   Opt[0]   Programmname aus PATH oder Programmpfad.
  //   Opt[1]   NULL oder Programmoption 1
  //   Opt[2]   NULL oder Programmoption 2
  //   ...
  //   Opt[n]   NULL! Optionsliste mit NULL abschließen!
```

Beispiel 1:

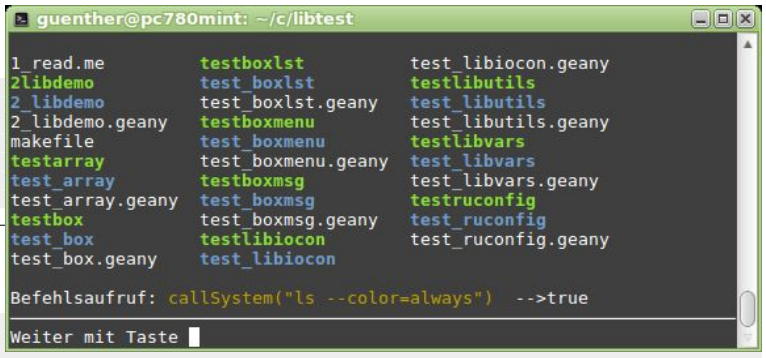
Die erste Zeile der Rückgabe von Befehl "uname -m" anzeigen.

```
  printf("CPU=%s", getPopenLine("uname -o"));
  ErrChkPrint(Err,"Fehler in getPopen()\n", false);
CPU=i686
```

Beispiel 2:

Rückgabe von Befehl "ls -l --color=always" in
Array mit Fehlerprüfung einlesen: getPopenArray(s, a)

```
  tArray *a=ArrayNew(0,0);
  s=tmpStr("ls -l --color=always");
  getPopenArray(s, a);
  ErrChkPrint(Err,"Fehler in getPopenLine()\n", false);
  ArrayPrint(a);
  ArrayDelete(a);
```



Beispiel 3:

Befehl "ls -l --color=always"
mit Fehlerprüfung aufrufen: callSystem(s)

```
  s=tmpStr("ls --color=always");
  printf
  ("\n"
   "Befehlsaufruf:"Fy" callSystem(\"%s\") "FN" -->%s\n"
   ,s
   ,BoolToStr( callSystem(s) )
  );
  ErrChkPrint(Err,"Fehler in callSystem()\n", false);
```

Beispiel 4:

Das laufende Programm durch ein neues ersetzen.

```
char *opt[4];                               // Programmoptionen
opt[0]="/home/pi/c/bin/chelp/bin/chelp";    // Programmname
opt[1]="-c";                                // Option
opt[2]="/home/pi/c/bin/infosys ";           // Option
opt[3]=NULL ;                               // Ende der Liste

callExitAndExec(Exit, opt, Debug);          // Exitfunktion "Exit"
```

oder

```
char *opt[]=
{ "/home/pi/c/bin/chelp/bin/chelp",    // Programmoptionen
  "-c",                                // Option
  "/home/pi/c/bin/infosys ",           // Option
  NULL                                 // Ende der Liste
}
callExitAndExec(Exit, opt, Debug);     // Exitfunktion "Exit"
```

Dateisystemfunktionen

Funktionen aus dem Modul files.h der Library utils.a .

Beschreibungen chelp: Stichworte/ Lib files.h: Files
Modul: c/lib/include/files.h
Testprogramm: c/libtest/testboxlist | s Test: ScanDir/ScanFiles Dateien und Ordner einlesen

```
/// ====================================================================
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// |
// | Library  : libutils.a | Dateisystemfunktionen
// | Modul    : files.h
// | Test/Doku: testboxlst, testlibutils, infosys
// |
// | Funktionen für Ordner und Dateien
// |  - Ordner- und Dateinamen lesen
// |  - Stringfunktionen für Dateinamen
// |  - Infos vom Dateisystem lesen
// |  - USB Datenträger mounten
```

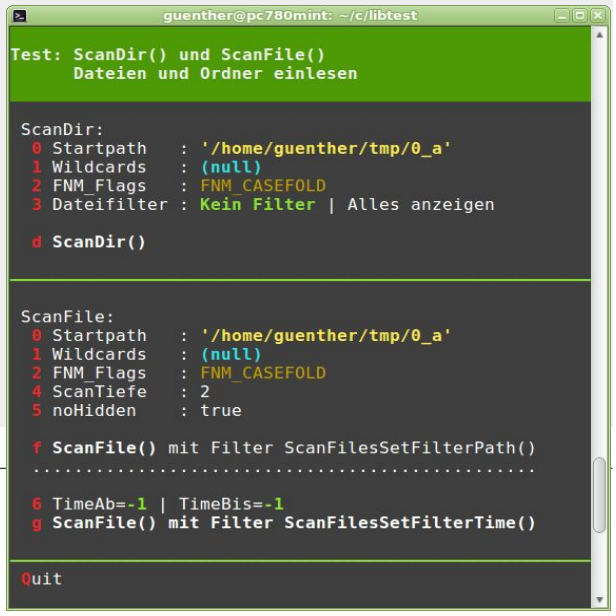
Die Funktion ScanDir liefert eine Liste von Ordnern, Files, Links und Devices **eines** Verzeichnisses.

- Die Rückgabe erfolgt mit einem String-Array.
- Die Auswahl erfolgt über vordefinierte oder eigenen Filterfunktionen.
- Filterfunktionen können Filetypen(d_type), Filterflags und Wildcards enthalten.
- Im Testprogramm können diese Filter getestet werden.

Der Dateidialog runBoxDirWahl() verwendet ScanDir.

```
// ====================================================================
// scanDir| ScanDir() liest Informationen über Files in ein Array ||||||||||||
//
bool ScanDir(const char *DirPfad, tArray *a, tFilter Filter) ;
  // Ordner- und Dateinamen aus DirPfad sortiert in das Array einlesen.
  // Filter  : NULL für DirFilterDefault oder eigener Dateifilter.
  // Rückgabe: true, Ordner- und Dateinamen im Array a mit 'struct dirent' Items. Sortiert mit alphasort. Ordner zuerst.
```

```
// Vordefinierte Filterfunktionen für ScanDir
//
 int DirFilterDefault(const struct dirent *p);
 // Filter: Alle sichtbaren Ordner, Files und "../" anzeigen.
 //         Verwendet keine Wildcards!

 int DirFilterFiles(const struct dirent *p);
 // Filter: Ordner, "../", sichtbare Files und Links anzeigen.
 //         Verwendet Wildcards mit DirFilterFlags.

 int DirFilterFilesH(const struct dirent *p);
 // Filter: Ordner, Links und Files anzeigen. Auch hidden!
 //         Verwendet Wildcards mit DirFilterFlags.

 int DirFilterDirs(const struct dirent *p);
 // Filter: Nur sichtbare Ordner und "../" anzeigen.
 //         Keine Wildcards und DirFilterFlags.

 int DirFilterNoDirs(const struct dirent *p);
 // Filter: Nur sichtbare Files und Links. Keine Ordner
 //         Verwendet Wildcards mit DirFilterFlags.
 ...
```



Die Funktion ScanFiles liefert eine Liste von den Ordnern, Files, Links und Devices eines ganzen **Verzeichnis-Baumes**.

- Die Rückgabe erfolgt mit einem String-Array.
- Die Auswahl erfolgt über vordefinierte oder eigene ScanFile-Filter.
- Im Testprogramm können diese Filter getestet werden.

```
// ====================================================================
// scanFiles| ScanFiles() liefert Datei-Infos zu Verzeichnisbäumen. ||||||||||
// Test/Doku: test_boxlst                                          ||||||||||
//
// Die Datei-Infos werden in einem Array vom Typ tScanFileInfo gespeichert.
// ScanFiles() verwendet das gesetzte ScanFiles-Filter zur Auswahl

tArray *ScanFiles(tScanFiles *ScanFiles, const char *StartDir, uint32_t Debug);
  // Das Rückgabe-Array mit ScanFileInfos zuerst löschen.
  // Alle Pfade im Verzeichnisbaum StartDir mit der Funktion nftw() bestimmen
  // und mit dem gesetzten ScanFiles-Filter prüfen. Für gültige Pfade
  // ScanFileInfos anlegen und im Rückgabe-Array speichern.
  // Das Rückgabe-Array wird von ScanFiles verwaltet.
  //
  // Eingaben: ScanFiles    Objekt mit Scan-Filter.
  //           StartDir     absolut oder relativ zum CWD.
  //           Debug>0      Einstellungen mit ScanFilesPrintInfo() anzeigen
  //
  // Rückgabe: Array vom Typ tScanFileInfo.
  //           NULL, wenn  ScanFiles==NULL oder relPath==NULL.
  //           Fehler im Err-Objekt.
  //
```

Stringfunktionen für Dateinamen:

Alle Rückgaben sind temporäre Strings.

Testprogramm: c/libtest/testlibutils | s Test: Stringfunktionen

```
const char *getProgPath();
  // Rückgabe: Programmpfad oder NULL mit Err

const char *getProgDir();
  // Rückgabe: Programmordner oder NULL mit Err

const char *getHomeDir();
  // Rückgabe: Homedirectory

const char *getCurrentDir();
  // Rückgabe: aktuelles Verzeichnis

const char *getTmpPath(const char *Praefix);
  // Rückgabe: Pfad für temporäre Dateien
  //           P_tmpdir/PraefixPID

void getRealDirAndBaseName(const char *Path, char **Dir,
                           char **BaseName);
  // Pfad mit realpath() auflösen.
  // '~' auf Pfad[0] wird zu Homeordner
  // Rückgabe:
  //    Dir      NULL oder tmpStr mit gültigem Ordner
  //    BaseName NULL oder tmpStr mit gültigem Dateinamen

const char *getFileName(const char *Path);
  // Rückgabe: NULL oder Filename ohne Dateiendung

const char *getDirName(const char *Path);
  //  Rückgabe: dirname()

const char *getBaseName(const char *Path);
  // Rückgabe: basename() mit Dateiendung

const char *getFileSuffix(const char*Path);
  // Rückgabe: NULL oder Dateierweiterung von Path

bool hasSuffix(const char *Path, const char *Suffixe);
  // Pfad    : Dateipfad
  // Suffixe : Gültige Extensions "*" oder z.B ".mp4 .txt ..."
  // Rückgabe: true, Pfad besitzt gültiges Suffix.

const char *getRealPath(const char *Path);
  // Rückgabe: absoluter Pfad. Links werden aufgelöst

bool isDir(const char *Path);
  // getFile| Überprüfen, ob Path ein Ordner ist.
  // Rückgabe: true, Path ist Directory. verwendet stat()

bool hasAccess(const char *Path, int W_Flags);
  // Prüft Zugriffsrechte W_Flags des Programms auf Path.
  // W_Flags: F_OK für Path existiert.
  //          R_OK, W_OK, X_OK  Zugriffsrechte.
  // Rückgabe: true/false. Fehler im Err-Objekt

... Funktionsliste unvollständig
```

```
guenther@pc780mint: ~
6 Test: Stringfunktion für Ordner- und Dateinamen

getProgPath(): /home/guenther/c/libtest/testlibutils
getHomeDir() : /home/guenther
getCWDir() : /home/guenther/c/libtest/test_libutils
getTmpPath("Praefix") : /tmp/Praefix4536

Gültige Dateipfade testen und zerlegen mit.
Funktion: getRealDirAndBaseName(Pfad, &Dir, &Basename)

Pfad: /home/guenther/c
      Eingabe :/home/guenther/c
      Dir     :/home/guenther/c
      Basename:(null)
      Ende mit ESC

Pfad: /home/guenther/c/

Rückgabe basename() mit Dateiendung. '~' auflösen.

getBaseName("/home/test.c"):  test.c
getBaseName("~/test.c"):  test.c
getBaseName("/home/test.bak.c"):  test.bak.c
getBaseName("/home/test"):  test
getBaseName("/home"):  home
getBaseName("/home/"):  (null)
getBaseName((null)):  (null)
getBaseName(""):  (null)
getBaseName("~"):  .
getBaseName("."):  .
getBaseName("./"):  (null)
getBaseName(".."):  ..
getBaseName("/"):  (null)
```

```
guenther@pc780mint: ~
Rückgabe dirname(). '~' auflösen.

getDirName("test.c"):  .
getDirName("~/test.c"):  /home/guenther
getDirName("~/abc/test.c"):  /home/guenther/abc
getDirName("/home/test.bak.c"):  /home
getDirName("/home/test"):  /home
getDirName("/home/test/"):  /home/test
getDirName((null)):  (null)
getDirName("/home/test."):  /home
getDirName("~"):  /home/guenther
getDirName("~/"):  /home/guenther
getDirName(""):  (null)
getDirName("./"):  .
getDirName("."):  .
getDirName(".."):  .
getDirName("/"):

Weiter mit Taste

isDir() existierend

isDir("/home"):  1
isDir("/home/"):  1
isDir("/home/x"):  0
isDir("/home/guenther/c"):  1
isDir("/home/guenther/c/libtest/1_linktest_ordner"):  1
isDir("/home/guenther/bin/chelp"):  0
isDir("/home/guenther/c/1_read.me"):  0
isDir("/home/guenther/c/libtest/lnk_1_read.me"):  0
isDir("/x"):  0
isDir("(null)"):  0

Weiter mit Taste
```

```
guenther@pc780mint: ~
getFileSuffix("/home/test.c"):  .c
getFileSuffix("/home/test.bak.c"):  .c
getFileSuffix("/home/test."):  .
getFileSuffix((null)):  (null)
getFileSuffix("/home/test"):  (null)
getFileSuffix("./test"):  (null)
getFileSuffix("."):  .
getFileSuffix(".."):  .

Weiter mit Taste

Mit getRealPath() können auch Links aufgelöst werden.

getRealPath("/home/pi"):  /home/guenther
getRealPath("~/"):  /home/guenther
getRealPath("~/c/1_read.me"):  /home/guenther/c/1_read.me
getRealPath("./"):  /home/guenther/c/libtest/test_libutils
getRealPath("../../"):  /home/guenther/c
getRealPath("/dev/disk/by-label/Old\x20Daten"):  (null)
getRealPath("/home/x"):  (null)
getRealPath((null)):  (null)

Weiter mit Taste
```

```
guenther@pc780mint: ~
getSubDirName("/home/guenther/","/home/guenther/tmp/"):  "tmp/"
getSubDirName("/home/guenther/","/home/guenther/"):  ""
getSubDirName("/home/guenther/","/home/"):  ""
getSubDirName("/home/guenther/","(null)"):  "/home/guenther/"
getSubDirName("(null)","/home/guenther/tmp/"):  ""
Weiter mit Taste
```

Hilfsfunktionen

Beschreibungen chelp: Stichworte/ Lib utils.h: Temporäre Strings, Hilfsfunktionen
Modul: c/lib/include/utils.h
Testprogramm: c/libtest/testlibutils

Alle Rückgaben sind temporäre Strings.

Zum Zerlegen von Strings in Teilstrings haben sich die Funktionen stripFirst() und stripItemI() bewährt.

Die fortlaufende Anwendung von stripFirst() zerlegt einen String durch Abtrennung des jeweils ersten Teilstrings mit einem Trennzeichen.

```
//=====================================================
// StrFnk| Stringfunktionen
// Alle Stringfunktionen liefern immer temporäre Strings.
```

```
char *stripFirst(const char *s, char Separator);
  // Zerlegt einen String mit Hilfe von Separatoren in
  // Teilstrings. Verwendet eine statische Kopie von s.
  // Kann nicht mit zwei Strings gleichzeitig arbeiten!
  //
  // Erster Aufruf mit String s!=NULL.
  // Rückgabe: 1. temp. Teilstring bis Separator oder Ende.
  //
  // Nachfolgende Aufrufe mit s=NULL:
  // Rückgabe: NULL oder nächster temp. Teilstring bis
  // Separator Ende.
  // NULL: Kein Teilstring mehr. Heapspeicher freigeben.
  //
  // Hinweis: Die Funktion belegt Heapspeicher. Er wird mit
  // dem letzten Teilstring freigegeben.
```

```
guenther@pc780mint: ~/c/libtest
Teilstrings mit stripFirst()

Code: teststrfnk.c
300 ▷   char  *s=NULL; char Sep='='; uint32_t i=0;
301 ▷
302 ▷   printf("Teststring: s=\"%s\", Sep='%c'  \n",s, Sep);
303 ▷
304 ▷   for (char *p=stripFirst(s, Sep); p!=NULL;  p=stripFirst(NULL, Sep), i++ )
305 ▷     printf( "%i: \"%s\"\n",i, p);
306 ▷   printLine(0);
307 ▷
308 ▷   s=" String a "; Sep='/';
309 ▷   printf( "Teststring: s=\"%s\", Sep='%c'  \n",s, Sep);
310 ▷   i=0;
311 ▷   for (char *p=stripFirst(s, Sep); p!=NULL;  p=stripFirst(NULL, Sep),i++ )
312 ▷     printf( "%i: \"%s\"\n",i, p);
313 ▷   printLine(0);
314 ▷
315 ▷   s=" String 1/ String 2/String 3   ";  Sep='/';
316 ▷
317 ▷   printf( "Teststring: s=\"%s\", Sep='%c'  \n",s, Sep);
318 ▷   i=0;
319 ▷   for (char *p=stripFirst(s, Sep); p!=NULL;  p=stripFirst(NULL, Sep),i++ )
320 ▷     printf( "%i: \"%s\"\n",i, p);
321 ▷   printLine(0);
322 ▷
323 ▷   s=" x= 365   "; Sep='=';
324 ▷   printf( "Teststring: s=\"%s\", Sep='%c'  \n",s, Sep);
325 ▷   i=0;
326 ▷   for (char *p=stripFirst(s, Sep); p!=NULL;  p=stripFirst(NULL, Sep),i++ )
327 ▷     printf( "%i: \"%s\"\n",i, p);

Teststring: s="(null)", Sep='='

Teststring: s=" String a ", Sep='/'
0: " String a "

Teststring: s=" String 1/ String 2/String 3   ", Sep='/'
0: " String 1"
1: " String 2"
2: "String 3   "

Teststring: s=" x= 365   ", Sep='='
0: " x"
1: " 365   "

Weiter mit Taste
```

Die zweite Variante stripItemI() ist günstiger, wenn auf die einzelnen Teilstrings in beliebiger Reihenfolge zugegriffen wird.

```
char *stripItemI(const char *s, int *StartI, char Separator, bool Trim);
  // Zerlegt einen String mit Hilfe von Separatoren in Teilstrings.
  //
  // s         : Eingabestring. Bleibt unverändert!
  // StartI    : Startindex für den nächsten Teilstring.
  // Separator: Trennzeichen.
  // Trim      : true, Teilstring trimmen.
  //
  // Rückgabe:
  //   NULL oder nächster temp. Teilstring bis Separator oder Stringende.
  //   StartI: Startindex für den nächsten Teilstring. -1: Kein Teilstring
  //           oder s==NULL oder StartI<0 oder StartI>=StrLen(s) .
```

```
guenther@pc780mint: ~/c/libtest
Test mehrfache Stringzerlegung mit stripItemI()

Code: teststrfnk.c
262 ▷   char Sep=','; char Sep1=' ';
263 ▷   char *s=newStr("  c5   f5  , c-dur 5 ");
264 ▷
265 ▷   printf("Der Eingabestring wird mit '%c' in Teilstrings zerlegt.\n",Sep);
266 ▷   printf("Jeder Teilstrings wird mit '%c' in Sub-Strings zerlegt und getrimmt.\n",Sep1);
267 ▷   printLn();
268 ▷   printf("Eingabestring: "FY"'%s'\n"FN, s);
269 ▷   printLn();
270 ▷
271 ▷   int i=0;
272 ▷   while (i>-1)
273 ▷   { char *t=stripItemI(s, &i, Sep,true);
274 ▷     printf(  " Teilstring   : "FG"'%s'"FN", i=%i\n",t ,i);
275 ▷
276 ▷     if (!t) continue;
277 ▷     int j=0;
278 ▷     while (j>-1)
279 ▷     { char *tt=stripItemI(t, &j, Sep1,true);
280 ▷       printf("    Sub-String: "FC"'%s'"FN", j=%i\n",tt ,j);
281 ▷     }
282 ▷   }
283 ▷   freePtr(s);

Der Eingabestring wird mit ',' in Teilstrings zerlegt.
Jeder Teilstrings wird mit ' ' in Sub-Strings zerlegt und getrimmt.

Eingabestring: '  c5   f5  , c-dur 5 '

 Teilstring   : 'c5   f5', i=12
    Sub-String: 'c5', j=3
    Sub-String: 'f5', j=-1
 Teilstring   : 'c-dur 5', i=-1
    Sub-String: 'c-dur', j=6
    Sub-String: '5', j=-1

Weiter mit Taste
```

Weitere Stringfunktionen:

```
const char *Str0(const char *s);          // Rückgabe: String nicht NULL! s==NULL liefert "".
const char *StrN(const char *s);          // Rückgabe: String nicht NULL! s==NULL liefert "NULL".
```

```
const char *StrToLower(const char *s);    // Rückgabe: tmpStr mit Kleinbuchstaben. UTF-8 Deutsch
const char *trimLeft(const char *s);      // Führende Blanks oder '\t' entfernen
const char *trimRight(const char *s);     // Blanks oder '\t' oder '\n' am Ende entfernen
const char *trimLR(const char *s);        // trimLeft und trimRight
const char *stripESCStr(const char*Str); // Esc-Sequenzen '\e...m' und '\e...K' entfernen
```

Weitere Stringfunktionen:

```
const char *getKeyValue(const char *Key, const char *String);
  // Input: Key und String mit Key="Wert" Paaren. Separatoren: ';' oder blanks.
  // Rückgabe: NULL oder Wert zum Key
  // Beispiel: char *s=" NAME=\"Daten 1\" TYPE=\"part\" RO=\"1\"";
  //           getKeyValue("TYPE", s) --> part
```

```
uint16_t countLines(const char *s);     // Rückgabe: Zeilenanzahl im String
const char *ESCStrToStr(const char*s);  // ESC-Sequenzen \n \e \t \k \\ \" in Bytes umwandeln . Rückgabe: tmpStr
void printESCStr(const char *s);        // Esc-Sequenzen in String anzeigen z.B. 1b5b43
```

Strings und Zahlen. Fehler im Err-Objekt.

```
int16_t  StrToInt16  (const char *s);
uint16_t StrToUInt16 (const char *s);           // Für Hexzahlen Prefix 0x
int32_t  StrToInt32  (const char *s);
uint32_t StrToUInt32 (const char *s);           // Für Hexzahlen Prefix 0x
uint32_t StrToUInt32b(const char *s, int Basis); // Zahlenbasis 2-36
double   StrToDouble (const char *s);           // String in Zahl umwandeln. Fehler im Err-Objekt
```

```
char *BytesToHexStr(const char *ByteStr);       // Bytefolge mit '\0' am Ende in Hexstring umwandeln.
char *BoolToStr(bool b);                        // Rückgabe: "true" oder "false"
```

```
char *ByteToBin(uint8_t Byte);           // Byte in formatierten Bin-String umwandeln . Rückgabe: Statische Strings.
char *WordToBin(uint16_t Word);          // Word in formatierten Bin-String '0b xxxx ...' umwandeln . Rückgabe: Statischer String.
uint16_t BinToWord(const char * Bin);    // Binärzahl '0b xxxx ...' in Word umwandeln
char *CHexStr(const char *s);            // String in Hex im Format für C-Compiler anzeigen. z.B. \x1b\x5b\x43
```

Stringlänge

```
int StrLen(const char *s);     // Input: String oder NULL. Verwendet strlen(). strlen(NULL) würde Fehler liefern.
int strlenUtf8(const char *s); // Input: String oder NULL. Rückgabe: Anzahl der UTF8 Zeichen.
```

Konvertierungen

```
char    *StrIsoNachUtf8(uint8_t *Bytes, uint32_t Len);   // Len ISO-8859-1 Bytes nach Utf8 String
uint8_t *StrUtf8NachIso(const char *Utf8, uint32_t Len); // Utf8 String nach ISO-8859-1 Bytes. Len mit StrLen(Utf8) berechnen.
```

Sichere Stringvergleiche auch mit NULL-Strings.

```
int   StrCmp(const char*a, const char *b);             // wie strcmp(),     case-sensitiv
int   StrNCmp(const char*a, const char *b, size_t n);  // wie strncmp)(),   case-sensitiv
int   StrCaseCmp(const char*a, const char *b);         // wie strcasecmp(), case-insensitiv
```

```
bool  isStrInStrS(const char *Haystack, const char *Needle);     // wie strstr():    true - Needle in Haystack. case-sensitiv.
bool  isStrInStr(const char *Haystack, const char *Needle);      // wie strcasestr(): true - Needle in Haystack. case-insensitiv.
bool  isStrFirstInStr(const char *Haystack, const char *Needle); // wie strstr(): true - Needle an 1. Stelle in Haystack. case-sensitiv.
char *CharInStr(const char *Haystack, char c);                   // wie index()
bool  isCharInStr(const char *Haystack, char c);                 // wie index()
```

Systemabfragen

```
bool isXRunning(); // true : GUI ist aktiv. Terminal emulator is running. false: Echte Console mit fixer Zeilenanzahl.
bool isPi();       // true : Raspberry Pi false: anderer Computer
bool isRoot();     // true : Rootrechte vorhanden
```

```
const char *getHostName();    // Hostname
const char *getUserName();    // Username
const char *getUserGroup();   // 1. Gruppe des Users
const char *getUserGroups();  // Gruppen des Users
```

```
const char *getTTYName();     // Name des Terminaldevices STDIN_FILENO ohne /dev/ . Rückgabe: pts/1 , tty1 . Fehler im Err-Objekt
const char *getSSHClient();   // NULL oder IP des SSH Client
const char *getShell();       // Shell l
const char *getTerminal();    // Terminal Bezeichner . Rückgaben: xterm, linux
const char *getSTY();         // screen Bezeichner
const char *getTermPath();    // Terminal PATH
```

```
pid_t getProgPid(const char*ProgName); // Pid mit "pidof -s ProgName" bestimmen. Rückgabe:  Pid einer(!) Programminstanz von ProgName
                                       // 0 Keine Programminstanz läuft. Löscht Err-Objekt!
uint32_t countProgs(const char*ProgName, uint16_t Debug); // Anzahl der Programminstanzen ermitteln. Rückgabe: Anzahl oder 0.
                                                          // Fehler im Err-Objekt. Löscht Err-Objekt!
```

```
bool isBigEndian();   // Bytefolge im Speicher des Systems. true : Big Endian: MSB,LSB | false: Little Endian: LSB,MSB
uint16_t BigToSysEndian(uint16_t BigEndian); Input: Byteorder Big Endian. Rückgabe: Systembyteorder
```

Mathematik

```
bool isZero(float x);            // float Vergleich mit Epsilonumgebung. Rückgabe: true: -EPSILON<=x<=EPSILON
      define EPSILON  0.00001    // Epsilonumgebung
float round1(float x);           // Runden auf eine Kommastelle. Linken mit -lm. makefile LIBS = -lm
```

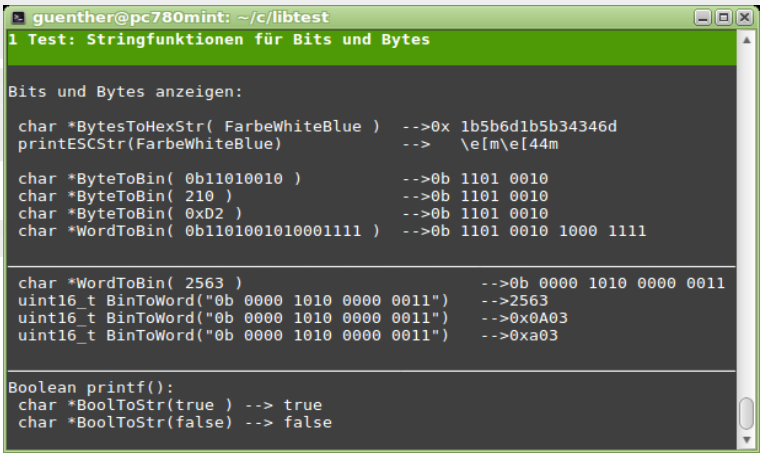
Bitweise vergleichen

```
bool isSet8 (uint8_t  PruefBits, uint8_t  Bits);
bool isSet16(uint16_t PruefBits, uint16_t Bits);
bool isSet  (int      PruefBits, int      Bits);
  // Rückgabe: true: Alle Prüfbits sind auch in Bits gesetzt
```

```
bool isOneBitSet8 (uint8_t  PruefBits, uint8_t  Bits);
bool isOneBitSet16(uint16_t PruefBits, uint16_t Bits);
bool isOneBitSet  (int      PruefBits, int      Bits);
  // Rückgabe: true: Mindestens ein Bit aus Prüfbits ist aucht in Bits gesetzt
```

Speicherstatus

```
void printHeapInfo();   // malloc_stats anzeigen
void printMallInfo();   // mallinfo() anzeigen
```



Library: vars.a

vars.a: Stellt globale Laufzeit-Variablen zur Verfügung.

Beschreibungen chelp:	Stichwort 'Lib vars.h : Var-Objekt'
Modul:	c/lib/include/vars.h
Testprogramme:	c/libtest/testlibvars | c/libtest/tesruconfig

Objekt Var

Das statische Objekt tVar dient zum Austausch und sicheren Speichern von globalen Daten.

- Die Laufzeit-Variablen werden bei der ersten Verwendung automatisch angelegt.
- Der Zugriff erfolgt über "Bezeichner", Pointer oder anonym mit VarNxtp().
- Zur besseren Übersicht können die Variablen in Gruppen/SubGruppen angelegt werden.
- Der Zugriff kann über die Gruppen/SubGruppen gefiltert werden.
- Übersichtliche Anzeige aller Variablen oder von Variablen Gruppen/SubGruppen.
- Einfaches Schreiben/Lesen der Variablen nach/von Konfigurationsdateien.
- Automatische Speicherverwaltung am Heap.

```
// ========================================================================
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// |
// | Library  : libvars.a | Globale Variablen. Konfigurationsdateien
// | Modul    : vars.h
// | Objekt   : tVar
// | Test/Doku: test_libvars, test_ruconfig
// |
// | Globale Variablenliste erzeugen und verwalten.
// | Konfigurations- und Scriptdateien schreiben/lesen (siehe Modul script.h).
```

Beispiel Var-Objekt:

Die Var-Werte können in Gruppen mit SubGruppen gespeichert werden.
Die Sichtbarkeit kann über Gruppen-Filter und SubGruppen-Filter eingestellt werden.

VarSetGrpFlt(GrpNr, SubGrpNr, GrpFilter, SubGrpFilter); GrpNr/SubGrpNr: int16_t Filter: true/false

Var-Variablen werden immer in der aktuellen Gruppe/SubGruppe angelegt: VarSetStr(Bezeichner , Wert);
Var-Werte können aus sichtbaren Gruppen/SubGruppen gelesen werden: VarGetStr(Bezeichner);

Für Bezeichner der Variablen sollten unbedingt im globalen Header mit #define Aliasse definiert werden.

```
z.B.  #define REMOTEHost        "RemoteHost"
```

Durch die Verwendung von Großbuchstaben am Anfang des Alias REMOTEHost können die Namen der Variablen im gesamten Projekt problemlos ersetzt werden! Der tatsächliche Bezeichner "RemoteHost" wird nur in der Konfigurationsdatei verwendet.

Beispiel:

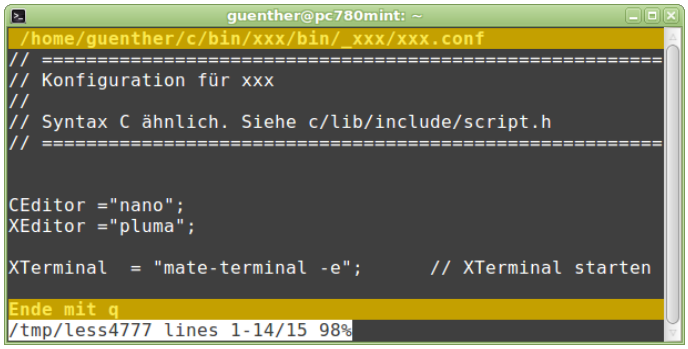
```
// ========================================================================
// Globaler Header
//
// ========================================================================
// Globale Konstanten
//
   define VERSION         "0.00"             // 2020-03-17 Programmversion
   define READMe          "1_read.me"        // Hilfedatei

   define CONFIGSuffix0   ".conf"            // Suffix Konfiguration
   define CONFIGSuffix    "*"CONFIGSuffix0   // Wildcard
   define CONFIGNameDef   "xxx"CONFIGSuffix0 // Default Konfiguration
   define ...
```

```
// ========================================================================
// Var-Objekt für globale Laufzeit-Variablen
//
  // Var-Bezeichner für Variablen aus dem Var-Objekt
  //
  // Var-Bezeichner Gruppe Programm ------------------------------------------
  // Diese Variablen werden 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 ...                         //
  //
  // Var-Bezeichner Gruppe Konfiguration -------------------------------------
  // Diese Variablen werden aus/in der Konfigurationsdatei gelesen/gespeichert
  //
   define CONFGrp      1              // Gruppe für die Konfiguration
   define CONFSubGrp   0              // Sub-Gruppe für die Konfiguration

   define XEDITOR      "XEditor"      // Var-Bezeichner für Editor für X
   define CEDITOR      "CEditor"      // Editor für Console
   define ...                         //
```

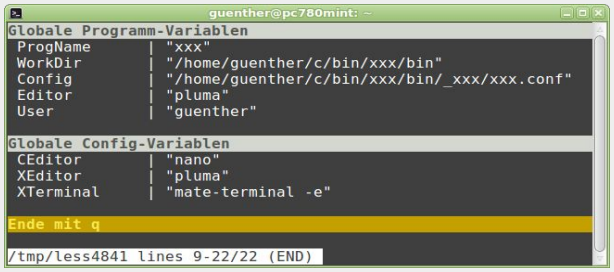
Beispiel Programmausschnitt: Init()

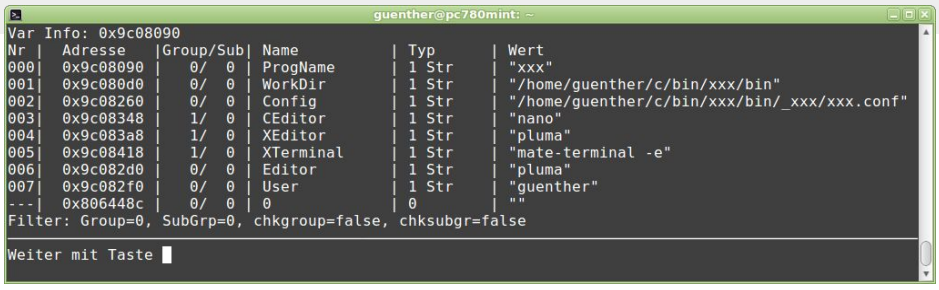
```
void Init()
{ // Variablen und Objekte initialisieren

  clrScr(); hideCursor();

  // Gruppe und Filter für Variablen setzen.
  VarSetGrpFlt(PROGGrp,0,false,false);  //Gruppen: PROGGrp/0, Filter: kein Filter

  // Den wirklichen Programmnamen und -dir bestimmen und setzen
  char *Dir, *BaseName;
  getRealDirAndBaseName( getProgPath(), &Dir, &BaseName );
  VarSetStr(PROGName, BaseName);
  VarSetStr(WORKDir,  Dir);

  ...
  chdir(Dir); // in Programmverzeichnis wechseln

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

  // Konfiguration lesen und Variablen in Var speichern
  if (!readConfig(VarGetStr(CONFIG), CONFGrp, Debug)) ErrExit("Fehler readConfig()\n");

  // Verwendeten Editor in PROGGrp/PROGSubGrp speichern
  VarSetGrpFlt(PROGGrp,PROGSubGrp,false,false);

  if (isXRunning()) VarSetStr(EDITOR, VarGetStr(XEDITOR)); // X Editor
  else              VarSetStr(EDITOR, VarGetStr(CEDITOR)); // Console Editor

  VarSetStr(USER, getUserName()); // Username
  ...
}
```

Die Konfigurationsdatei:




```
Beispiel Variablen anzeigen:

void printVar()
{ // Variablen geordnet anzeigen
  clrScr();

  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();
  WeiterMitTaste();

  VarPrint();  // Alle Var's anzeigen
  printLine(0);
  WeiterMitTaste();
}
```





Beispiel: Globale Strukturvariablen

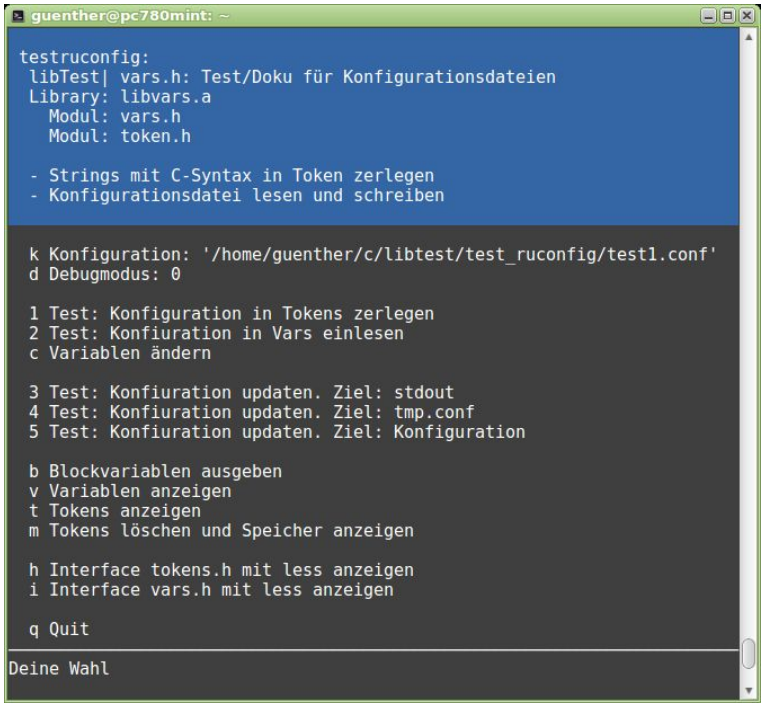
Beschreibungen chelp:	Stichwort 'Lib vars.h: Var-Objekt | g| Struktur Var |vars.h| Struct-'
Modul:	c/lib/include/vars.h
Testprogramme:	c/libtest/testlibvars | 3 Test: Struct-Variablen anlegen/löschen

Strings mit C-Syntax zerlegen.

Objekt tTokens kann Strings mit sehr einfacher C-Syntax in eine Tokenliste zerlegen.
Das Objekt wird zum Lesen von Konfigurationsdateien, Scriptdateien und anderen Dateiformaten verwendet.

Beispiel: Stichwortliste aus chelp, Funktion KeywordsParseDatei().

```
// ================================================================================
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// |
// | Library  : libvars.a | Text mit C-Syntax in Token zerlegen
// | Modul    : token.h
// | Objekt   : tTokens
// | Test/Doku: testruconfig
// |
// | Objekt tTokens kann Strings mit sehr einfacher C-Syntax
// | in eine Tokenliste zerlegen.
// |
// | Variablen und Blöcke {...} der Tokenliste können im globalen
// | Var-Objekt abgespeichert werden.
// |
// | Variablen und Blöcke {...} der Tokenliste können mit den Variablen aus
// | dem Var-Objekt aktualisiert und ausgegeben werden.//
```



Modul: token.h
Testprogramm: c/libtest/testruconfig

Beispiel Test 1:

Test1 Liest die Konfigurationsdatei test1.conf.

Die Bildschirmkopie links zeigt folgenden Ausschnitt der Datei test1.conf:

```
...
// ====================================================

Variable ="Test";
str1     ="neu";  // wird im Test geändert

// Stringvariablen:
VarSetGroups(9,1);
Block1[] = {
 15,
 21,

...
```

Konfigurationsdateien

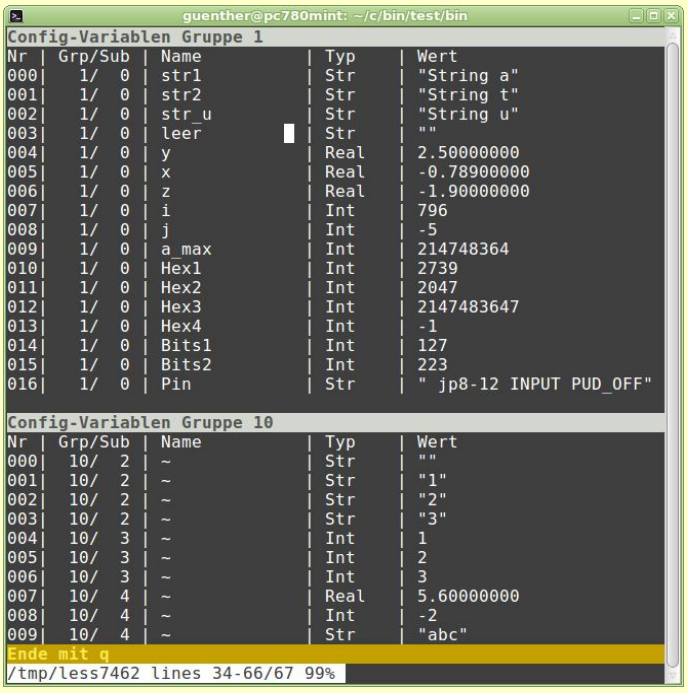
Modul: c/lib/include/script.h
Testprogramm: c/libtest/testruconfig

```
// ============================================================================
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// |
// | Library  : libvars.a | Konfigurations- und Scriptdateien mit Bashbefehlen
// | Modul    : script.h
// | Objekt   : tTokens
// | Test/Doku: testruconfig, Konfigurationsdatei suchen, lesen oder updaten
// |
// | Test/Doku: testscript, Scriptdateien lesen und ausführen
// |
// |
// ============================================================================
// Alle Interface-Header sind in $(HEADER_DIR)
```

Beispiel einer Konfigurationsdatei:

Es wird eine sehr einfache C-Syntax verwendet. Die Verarbeitung erfolgt mit Objekt tToken. Anzeige mit VarPrintMitFilter(false);

```
// ================================================
// Konfigurationsdatei mit Variablen und Datenblöcken.
//
// Kommentare :  nach //
// Variablen  :  Variable = Wert;
// Datenblöcke:  Block[]={ Wert1, Wert2, Wert3, ... }
// ================================================

// Stringvariablen: Name = "..."  ---------------------
 str1="String a";  // Kommentar
 str2="String t";  str_u="String u";  leer="";

// Reelle Zahlen mit Dezimalpunkt '.' -----------------
 y = 2.50000000; x = -0.78900000; z=-1.90000000;

// Integer vom Typ int32_t von INT32_MIN bis INT32_MAX
  i=796; j=-5; a_max=214748364;

// Hex Integer Eingabe --------------------------------
 Hex1  = 0xAB3; // hex, 0x... Keine Blanks verwenden
 Hex2  = 2047;  Hex3  = 2147483647;  Hex4  = -1;

// Binär Integer Eingabe ------------------------------
 Bits1 = 127;
 Bits2 = 0b 1101 1111; // binär, 0b... Banks möglich

// Beispielstring zum Zerlegen ------------------------
 Pin=" jp8-12 INPUT PUD_OFF";

// Grp/SubGrp -----------------------------------------
// Der Befehl VarSetGroups(n, m) schaltet auf
// Gruppe n,  SubGrp m


// Blöcke ---------------------------------------------
// Block[]={ } definiert Blöcke für fortlaufende Variablen
// Kommentare in Blöcken bleiben beim Update nicht erhalten!

// Variablen in Guppen definieren
 VarSetGroups(10, 2); Block[]={"", "1", "2" ,"3" };
 VarSetGroups(10, 3); Block[]={1, 2, 3 };
 VarSetGroups(10, 4); Block[]={ 5.60000000,  -2,  "abc" };
```



Konfigurationsdatei suchen:

```
const char *findConfigPfad(const char *ProgName, const char *ConfigName);
  // Konfiguration suchen
  //   1. Versuch : ~/.config/ProgName/ConfigName
  //   2. Versuch : ./_ProgName/ConfigName
  // Rückgabe: Pfad oder NULL mit Fehler in Err.
```

Konfiguration in die globalen Variablen einlesen:

```
bool readConfig( const char *Pfad, uint16_t Group, uint16_t Debug);
  // Konfigurationsdatei Pfad lesen und die Variablen und Blöcke {..} in Var
  // speichern. Fehlermeldungen anzeigen.
  // Group: Startwert für Var Gruppe/Filter ist VarSetGrpFlt(Group, 0, true, false).
  // Debug: 0-2. 1: Konfiguration zeilenweise anzeigen.
  //
  // Kommentare: // ... Kommentare bleiben beim Speichern erhalten. Ausnahme: Blöcke.
  //
  // Wertzuweisungen:  Name=Wert;
  //                   Der Wert wird unter "Name" in Var in der aktuellen
  //                   SubGrp von Group gespeichert. Start mit SubGrp=0.
  //
  // Stringverkettung: s = a + "..."  oder s = a "..."
  //
  // Blöcke: Block[]={ Wert1,  23,  -0.50  ,0b1110111, 0xFA, "Text" ... }
  //         Blöcke bestehen aus durch ',' getrennten Werten. Diese Werte werden
  //         als anonyme Vars in der aktuellen Group/SubGrp gespeichert.
  //         Kommentare in Blöcken gehen verloren.
  //
  // Funktionen: Gruppen in der Konfiguration setzen:
  //             VarSetGrpFlt(Gruppe,SubGruppe, ChkGrp, ChkSubGrp); Parameteranzahl von 1-4;
  //             VarSetGrpFlt("Script") ruft die Funktion ScriptSetGrps();
  //
  // Rückgabe: false, Konfiguration nicht oder unvollständig gelesen.
```

Die Werte von globalen Variablen aus der Konfiguration updaten:

Sonstige Variablen der Konfiguration bleiben unverändert!

```
bool updateConfig( const char *Pfad, uint16_t Group, uint16_t ZielModus, uint16_t Debug);
  // Konfigurationsdatei Pfad lesen und updaten. Fehlermeldung anzeigen.
  //
  // Pfad:      Konfigurationsdatei
  // Group:     Startwert für Variablengruppe. VarFilter ist true/true.
  // ZielModus: legt das Ausgabeziel fest.
  //            0: stdout
  //            1: tmp.cfg
  //            2: Pfad
  // Debug:     Debugmodus 0..2. 0 keine Infos anzeigen
  //
  // Die Variablen und Blöcke {..} der Konfigurationsdatei werden
  // mit den Werten aus Var aktualisiert.
  //
  // Wertzuweisungen: Name=Wert;
  //                  Wenn Var "Name" im aktuellen Var-Filter existiert, dann wird der
  //                  Wert in der Datei aktualisiert. Kommentare bleiben erhalten.
  //
  // Funktionen: Gruppen in der Konfiguration setzen/ändern:
  //             VarSetGrpFlt(Gruppe,SubGruppe, ChkGrp, ChkSubGrp); Parameteranzahl 1-4;
  //             VarSetGrpFlt("Script") ruft die Funktion ScriptSetGrps();
  //
  // Blöcke: Block[]={ wert1, wert1, ... };
  //         Alle anonymen Werte aus der aktuellen Group/SubGrp von Var werden
  //         als Block { ... } in die Zieldatei Ziel geschrieben. Die Originalwerte
  //         der Konfigurationsdatei und Kommentare innerhalb von { } gehen verloren.
  //
  // Rückgabe: false für Fehler
// ============================================================================
```

Beispiel: Objekt Array

```
// ========================================================================
// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
// |
// | Library  : libvars.a | Arraycontainer für Strings und sonstige Objekte
// | Modul    : array.h
// | Objekt   : tArray
// | Test/Doku: testarray
// |
// | tArray ist ein Container zum indizierten Zugriff auf beliebige Datenitems.
// |  - Der Daten-Zugriff kann über uint32_t Indizes oder Pointer erfolgen.
// |  - Freie Anfangsgröße und automatische Erweiterung.
// |  - tArray kann Daten neu anlegen oder bestehende Daten übernehmen.
// |  - Alle Funktionsaufrufe erwarten einen Arraypointer ungleich NULL
```

Strukturarrays

Das Objekt tArray kann als Container für beliebige Datenstrukturen verwendet werden.
Die Defaulteinstellungen liefern ein String-Array. Für beliebige Strukturen können folgende Funktionen gesetzt werden:

Deletefunktionen für ein Item:

```
          Funktionstyp:   typedef void (*tArrayDeleteItem) (void *Item);  // für eigene Strukturen
          Vordefiniert:   void ArrayDeleteItemFree(void *Item)            // Default: Item mit freePtr() freigeben
                          void ArrayDeleteItemNop(void *Item)             // keine Aktion
```

Printfunktion für ein Item:

```
          Funktionstyp:   typedef const char * (*tArrayItemStr) (const void *Item);  // für eigene Strukturen
          Vordefiniert:   const char *ArrayItemStr(const void * Item)                // Default: Bytes bis zum ersten '\0' ausgeben.
                          const char *ArrayItemPtrStr(const void * Item)             // Adresse des Datenitems
```

Zum Suchen und Sortieren:

```
          Funktionstyp:   typedef int (*tArrayCompare) (const void *a, const void *b); // für eigene Strukturen
          Vordefiniert:   int ArrayCompareStr(const void *Item1, const void *Item2)    // Stringvergleich mit strcmp()
```

Modul: c/lib/include/vars.h
Programm: c/libtest/testarray

Beispiel: Sortiertes Array für konstante Pointer

```
  ...

  tArray *a=ArrayNew(10,0);
  ArraySetDelete(a, ArrayDeleteItemNop);  // keine Freigabe
  ...

  ArrayAddPtr(a,"xxx");
  ArrayAddPtr(a,"iii");
  ArrayAddPtr(a,"ccc");
  ArrayAddPtr(a,"aaa");
  ArrayAddPtr(a,"fff");
  ArrayAddPtr(a,"eee");

  for (uint32_t i=0; i<ArraySize(a); i++)
    printf("%3i -> \"%s\"\n", i, ArrayPrintItem(a,i));

  ArrayQSort(a, NULL);   // Quicksort mit ArrayCompareStr
  ArrayPrint(a);         // Default
  ...

  a=ArrayDelete(a);
```

```
guenther@pc780mint: ~/c/libtest
4 Test: Pointer auf bestehende Daten in Array übernehmen.
        Daten sortieren.
        Ein Beispiel mit Stringkonstanten.

Array anlegen: tArray *a=ArrayNew(10,0)
Pointer in Delete nicht freigeben:
Destruktor Nop  ArraySetDelete(a, ArrayDeleteItemNop) setzen.

Pointer auf bestehende Daten im Array speichern:
Die Pointer-Daten werden nicht kopiert!
Ein Beispiel mit Stringkonstanten.
  ArrayAddPtr(a,"xxx")
  ArrayAddPtr(a, ...) usw.

  0 -> "xxx"
  1 -> "iii"
  2 -> "ccc"
  3 -> "aaa"
  4 -> "fff"
  5 -> "eee"

Daten sortieren: ArrayQSort(a,cmpString)
Vergleichsfunktion: int cmpString(const void *a, const void *b)
Anzeige: ArrayPrint(a)

aaa
ccc
eee
fff
iii
xxx

Freigabe:

Array mit ArrayDelete(a) freigeben. Der eingestellte
Datendestruktor ArrayDeleteItemNop() gibt die Datenitems
nicht frei!.
```

Beispiel: Strukturarray

```
typedef struct tPunkt                   // Datenstruktur
 { int32_t x;
   int32_t y;
 } tPunkt;

const char *PunktStr(const void *p)     // Printfunktion
 { return tmpStrF( "Punkt x=%i, y=%i"
                  ,((tPunkt *) p)->x
                  ,((tPunkt *) p)->y );
 }
```

```
void Test1()
{ ...
  tArray *a=ArrayNew(0,0);              // Array anlegen
  ...

  ArrayAdd(a                            // Daten hinzufügen
           ,&(tPunkt){.x=1, .y=12 }
           ,sizeof(tPunkt)
          );
  ...

  ArraySetPrint(a,PunktStr);            // Printfunktion setzen
  ArrayPrint(a);

  a=ArrayDelete(a);
  ...
}
```

```
guenther@pc780mint: ~/c/libtest
1 Test: Struct-Array anlegen, Structs speichern und löschen.
        ArrayNew(0,0);

Array anlegen: tArray *a=ArrayNew(0,0)
Verzeichnisgröße default, VerzIncr default

Datenstruktur tPunkt:
typedef struct
 { int32_t x;
   int32_t y;
 } tPunkt;
 ItemStr-Funktion für ArrayPrint():
const char *PunktStr(const void *p)

Punkte hinzufügen:
 ArrayAdd(a, &(tPunkt){.x=1, .y=2}, sizeof(tPunkt));
 ...

Array-Daten anzeigen:
ItemStr-Funktion setzen: ArraySetPrint(a,PunktStr)
Anzeigen: ArrayPrint(a)

Punkt x=1, y=12
Punkt x=2, y=23
Punkt x=2, y=80

Array und Daten löschen: ArrayDelete(a)
```

Array speichern

Die Datenstrukturen von Arrays können auch in strukturierten Text-Dateien gespeicher oder von Text-Dateien gelesen werden.

Beispiel: Siehe Testprogramm c/libtest/test_rwstruct.

Beispiel einer Text-Datei mit Kommentar und Datenstrukturen vom Typ tEvent in einfacher C-Syntax.

```
//Array mit Items vom Typ tEvent
Events[]=
{{Bez ="Temp 1",
  Typ ="t",
  n   =-95,
  Grad=-20.500000,
 },
 {Bez ="Relay A",
  Typ ="r",
  n   =6,
  Grad=0.250000,
 },
}
```

Die Definitionen zu obiger Datei:

Die Strukturdefinition von Typ tEvent für das Struktur-Array:

```
// Arraydefinition =============================================================
// |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
//
  // ...........................................................
  // Deklaration eines Arrayitems vom Typ tEvent:

  void        EventDelItem(void *Item);        //Löschfunktion fürs Item
  const char *EventPrtItem(const void *Item);  //Printfunktion fürs Item

  typedef struct tEvent   // Beispiel einer Eventbeschreibung
  { char       *Bez;      // Eventbezeichner
    char        Typ;      // Eventtyp
    int32_t     n;        // Index je nach Typ. z.B. Sensorindex
    double      Grad;     // Temperatur
    uint16_t    Debug;    // >0 Debugmodus für das einzelne Event
  } tEvent;
```

Zur Datenstruktur tEvent wird eine Beschreibung der Datenspeicherung definiert: Struktur tDataDef.
Für einen Feldbezeichener .FeldBez wird der C-Datentyp .DatTyp und der File-Datentyp .TokTyp definiert.
Die Reihenfolge und Anzahl in der Feldbezeichner kann in der Datendefinition tDataDef kann frei gewählt.

Im Beispiel wird das Feld Debug nicht im File gespeichert.

```
  // ...........................................................
  //
  // Datendefinition für das Array 'Events':
  // Die Definition beschreibt die Daten zum Schreiben und Lesen.
  // DatTyp ist der im C-Programm (frei) definierte Datentyp.
  // TokTyp ist der Datentyp aus 'tokens.h' für die Speicherung in der Datei
  //
  tDataDef EventDef=           // Datendefinition für Array 'Events'
  {.a=NULL,                    // Arraypointer wird für read/write gesetzt
    //                         // NULL für alle weiteren DatenDef's in '.StructDef'
   .aDel=EventDelItem,         // Deletefunktion fürs Array
   .aPrt=EventPrtItem,         // Printfunktion fürs Array
   .Bez="Events",              // NULL oder Bezeichner fürs Datenfile
   .StructSize=sizeof(tEvent), // Byteanzahl des Items tEvent
   .StructFeldAnz=0,           // Feldanzahl im Item. Immer 0 setzen.
   //                          // Der Wert wird aus 'StructDef' berechnet!

   .StructDef=                 // Feldbeschreibungen für struct tEvent von Array 'Events'
   {// Feldname        | C-Datentyp    | Feld-Offset in struct          | Datentyp im File
     {.FeldBez="Bez",   .DatTyp=STRING, .Offset=offsetof(tEvent, Bez)  , .TokTyp=String  },
     {.FeldBez="Typ",   .DatTyp=CHAR,   .Offset=offsetof(tEvent, Typ)  , .TokTyp=String  },
     {.FeldBez="n",     .DatTyp=INT32,  .Offset=offsetof(tEvent, n)    , .TokTyp=Integer },
     {.FeldBez="Grad",  .DatTyp=REAL,   .Offset=offsetof(tEvent, Grad) , .TokTyp=Real    },
     {.FeldBez=NULL } // NULL: Ende der Definition
    },
  };
```

Eine Funktion zum Lesen oder Schreiben der Array-Strukturen kann dann wie folgt dekariert werden:

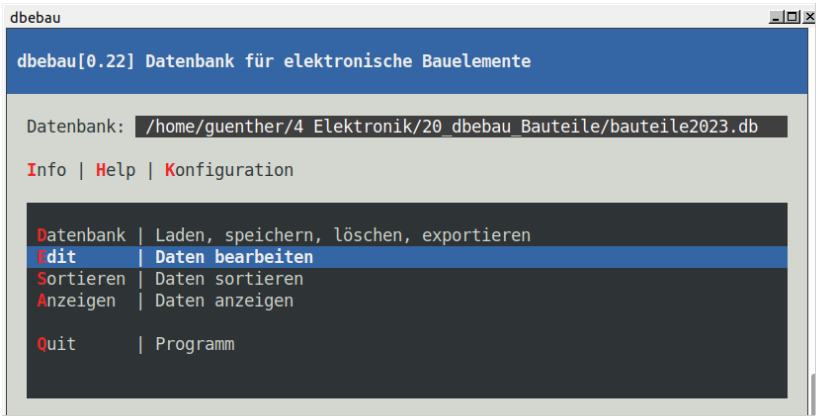
```
// Arraydaten write/read ---------------------------------------------------
//
// Array mit Items vom Typ tEvent schreiben oder lesen.
// Die Richtung wird von WR bestimmt: 'r'ead oder 'w'rite
//
void ArrayStructWR(const char *DatPath, tDataDef *DatDef, char WR)
{
  DataOpen(DatPath, WR);                      // Open Datei im Modus 'r' oder 'w'
  DataWRem("Array mit Items vom Typ tEvent"); // Im Modus 'w' Remark schreiben
  DataArrayWR(EventDef);                      // Im Modus 'r' read oder Modus 'w' write
  DataChkErr(false);                          // Fehlerausgabe
  DataClose();                                // Close Datei
} // ----------------------------------------------------------------------
```

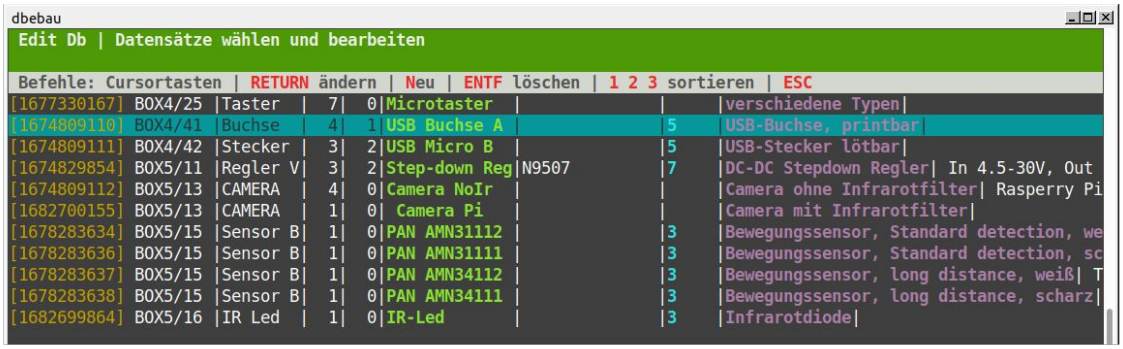
Beispiel dbebau

Beispiel: Datenbank für elektronische Bauelemente

Das Programm dbebau verwendet eine Arraystruktur zum Speichern der Bauelemente.

Die Implementierung der Datebank findet man im Modul db.h und db.c.





Datei bauteile2023.db:

```
// Datenbank für elektronische Bauteile
// Datenformat für Programm 'dbebau'
// Speicherdatum: 20240619_212048

Bauteile[]=
{{Id="..."
  ...
 },
...
 {Id="20230127_094510",
  Lager="BOX4/41",
  Typ="Buchse",
  Bezeichner="USB Buchse A",
  Aufdruck=" ",
  Package="5",
  Funktion="USB-Buchse, printbar",
  Anzahl=4,
  Used=1,
  Infos=" ",
 },
 {Id="20230127_094511",
  Lager="BOX4/42",
  Typ="Stecker",
  Bezeichner="USB Micro B",
  Aufdruck=" ",
  Package="5",
  Funktion="USB-Stecker lötbar",
  Anzahl=3,
  Used=2,
  Infos=" ",
},
 { ...
 },
}
```

GNU General Public License

```
/*
 *
 * Copyright 2022-2025 Günther Schardinger <schardinger@projektc.at>
 *
 * 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.
*/
```