

Modernes C

Praktikum „C-Programmierung“



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Eugen Betke, Nathanael Hübbe, Michael Kuhn, Jakob Lüttgau, Jannek Squar

2020-01-20

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

Modernes C

Einführung

Beispiele

GLib

Zusammenfassung

- Erinnerung: In den 1970ern von Dennis Ritchie entwickelt
 - Seitdem stetige Weiterentwicklung
- Funktionsumfang wird durch Standards festgelegt
 - Bis 1989: Buch “The C Programming Language” von Brian W. Kernighan und Dennis Ritchie als Quasi-Standard (K&R C)
 - Ab 1989: Standardisierung durch ANSI (ANSI C, C89)
 - Ab 1990: Internationale Norm durch ISO (C90, entspricht C89)
 - Ab 1995: Erste Erweiterung (C95)

- Ab 1999: Neuer Standard (C99)
 - Größtenteils mit C90 kompatibel
 - Neue Funktionen, teilweise von C++ übernommen
 - Echte Booleans, komplexe Zahlen etc.
- Ab 2011: Neuer Standard C11
 - Neue Funktionen und bessere Kompatibilität mit C++
 - Atomare Datentypen, Threads etc.
- Ab 2018: Neuer Standard C18 (entspricht C11 plus Fehlerkorrekturen)

- Fehlerbehandlung kann in C recht aufwendig sein
 - Keine Unterstützung für Exceptions etc.
- Häufiges Muster:
 - Eine Funktion soll vorzeitig verlassen werden
 - Allokierte Variablen müssen aufgeräumt werden etc.
- Grundsätzlich zwei Möglichkeiten
 - Code-Duplikation in jedem Fehlerfall
 - goto-Marke für Fehlerfälle

```
1 void foo (char* bar) {
2     char* foo = malloc(1024);
3     if (bar == NULL) {
4         free(foo);
5         return;
6     }
7     free(foo);
8 }
9
10 int main (void) {
11     foo(NULL);
12     return 0;
13 }
```

- free muss in jeder Fehlerabfrage wiederholt werden

```
1 void foo (char* bar) {
2     char* foo = malloc(1024);
3     if (bar == NULL)
4         goto error;
5 error:
6     free(foo);
7 }
8
9 int main (void) {
10     foo(NULL);
11     return 0;
12 }
```

- goto erlaubt das Anspringen einer speziellen Fehlerbehandlung
 - In diesem Fall identisch zum normalen Funktionsende

```
1 void cleanup_free (void* p) { free(*((void**)p)); }
2
3 void foo (char* bar) {
4     __attribute__((cleanup(cleanup_free))) char* foo = malloc(1024);
5     if (bar == NULL)
6         return;
7 }
8
9 int main (void) {
10     foo(NULL);
11     return 0;
12 }
```

- `cleanup`-Attribut erlaubt das Aufräumen am Ende des Gültigkeitsbereichs
 - Nicht standardisiert, wird aber von GCC und Clang unterstützt

```
1 void foo (char* bar) {
2     g_autofree char* foo = malloc(1024);
3     if (bar == NULL)
4         return;
5 }
6
7 int main (void) {
8     foo(NULL);
9     return 0;
10 }
```

- GLib macht die Benutzung komfortabler
 - Außerdem weitere g_auto-Varianten für beliebige Datentypen

```
1 int foo = 0;
2 int thrd (void* bar) { for (int i = 0; i < 10000; i++) foo++; }
3
4 int main (void) {
5     thrd_t thrds[10];
6     for (int i = 0; i < 10; i++) thrd_create(&thrds[i], thrd, NULL);
7     for (int i = 0; i < 10; i++) thrd_join(thrds[i], NULL);
8     printf("%d\n", foo);
9     return 0;
10 }
```

- Threads erlauben die parallele Abarbeitung von Aufgaben
 - Sowohl zur Leistungssteigerung als auch zur Programmstrukturierung
- In diesem Fall unkoordinierter Zugriff auf gemeinsame Variable

```
1 atomic_int foo = 0;
2 int thrd (void* bar) { for (int i = 0; i < 10000; i++) foo++; }
3
4 int main (void) {
5     thrd_t thrds[10];
6     for (int i = 0; i < 10; i++) thrd_create(&thrds[i], thrd, NULL);
7     for (int i = 0; i < 10; i++) thrd_join(thrds[i], NULL);
8     printf("%d\n", foo);
9     return 0;
10 }
```

- Atomare Variablen ermöglichen koordinierten Zugriff mit geringem Overhead
 - Bisher nur mit speziellen Funktionen möglich

- Einige Funktionen sind unbequem zu benutzen
 - Z. B. das automatische Aufräumen von Variablen
- Es ist schwierig portablen Code zu schreiben
 - Eigenheiten von Linux, macOS und Windows
- Selbst implementierte Datentypen und Algorithmen sind oft fehleranfällig
 - Z. B. verkettete Listen, Bäume etc.
- GLib ist eine häufig genutzte C-Bibliothek
 - *GLib is a general-purpose utility library, which provides many **useful data types, macros, type conversions, string utilities, file utilities, a mainloop abstraction, and so on.** It works on many UNIX-like platforms, as well as Windows and OS X. GLib is released under the GNU Lesser General Public License (GNU LGPL). [1]*

```
1 int main (void) {  
2     g_autoptr(GString) string = g_string_new(" ");  
3     g_string_prepend(string, "Hallo");  
4     g_string_append(string, "Welt!");  
5     g_print("%s\n", string->str);  
6     return 0;  
7 }
```

- GString-Datentyp erlaubt das komfortable Arbeiten mit Strings
 - Speicher wird transparent durch GLib verwaltet
 - Zugriff auf C-String weiterhin möglich

```
1 void bar (int* foo) {
2     g_rc_box_acquire(foo);
3     *foo += 42;
4     g_rc_box_release(foo);
5 }
6
7 int main (void) {
8     int* foo = g_rc_box_new0(int);
9     bar(foo);
10    g_rc_box_release(foo);
11    return 0;
12 }
```

- Erlaubt es Datentypen nachträglich mit Reference Counting auszustatten
 - Native Unterstützung mit grefcount und gatomicrefcount

```
1 void foo (GError** error) {
2     g_return_if_fail(error == NULL || *error == NULL);
3     g_set_error(error, 1, 0, "Adresse: %p", error);
4 }
5
6 int main (void) {
7     GError* error = NULL;
8     foo(&error);
9     if (error != NULL)
10         g_print("Fehler: %s\n", error->message);
11     return 0;
12 }
```

- GError stellt ein Framework zur Fehlerbehandlung bereit
 - Kontrolle über Rückgabe liegt bei Aufrufendem

- GLib bietet außerdem eine Vielzahl weiterer Datentypen und Algorithmen
 - Threads, Thread-Pools, Allokatoren etc.
 - String-, Zeit-, Datei- und Parser-Funktionen
 - Listen, Bäume, Queues, Arrays etc.
- Zusätzliche Infrastruktur
 - Test- und Logging-Frameworks

- Neuere C-Standards bieten moderne Funktionalität
 - Threads, atomare Datentypen, Unicode, statische Asserts etc.
- GLib erleichtert die Nutzung von C
 - Portabilität, Komfort und Sicherheit

- [1] The GNOME Project. **GLib Reference Manual**.
<https://developer.gnome.org/glib/>.
- [2] Wikipedia. **C (Programmiersprache)**.
[https://de.wikipedia.org/wiki/C_\(Programmiersprache\)](https://de.wikipedia.org/wiki/C_(Programmiersprache)).