Praktikum C-Programmierung 2026

Modulares Programmieren

Jannek Squar

2026-06-10

Scientific Computing
Universität Hamburg

Outline

Funktionspointer

Externer Code

- Header
- Bibliotheken
- Suchpfade

Table of Contents

Motivation

- Bisher: Pointer zeigt auf Daten
- Funktionspointer [1]: Pointer zeigt auf Funktion
- Callback-Funktionen, Dynamische Funktionsaufrufe
- Deklaration: `<Rückgabetyp> (*<Funktionspointer-Name>)(<Parameterliste>)`

```c
#include <stdio.h>

int add(int a, int b) {
  return a + b;
}

int main() {
  int (*fptr)(int, int);
  fptr = &add; // & optional
  printf("add: %p\n", add);
  printf("fptr: %p\n", fptr);
  printf("Ergebnis: %d\n", fptr(10, 5));
  return 0;
}
```

```
$ ./example_add.x
add: 0x400466
fptr: 0x400466
Ergebnis: 15
```

Beispiel Callback

```c
#include <stdio.h>

int add(int a, int b) {
  return a + b;
}

int subtract(int a, int b) {
  return a - b;
}

void calc(int a, int b, int (*op)(int, int)) {
  printf("%d\n", op(a, b));
}

int main() {
  calc(10, 5, add);
  calc(10, 5, subtract);
  return 0;
}
```

```
./example_calc.x
15
5
```

Beispiel Callback II

Beispiel pthreads

```
1  #include <pthread.h>
2
3  int pthread_create(pthread_t *thread,
4                     const pthread_attr_t *attr,
5                     void *(*start_routine)(void*),
6                     void *arg);
```

Beispiel Callback III [8]

```c
#include <pthread.h>
#include <stdio.h>

void *foo(void *arg) {
  printf("%s\n", (char *)arg);
  return NULL;
}

int main() {
  pthread_t thread1, thread2;
  void *(*bar)(void *) = foo;

  pthread_create(&thread1, NULL, foo, (void *)"Thread1 is
  ↪  running.");
  pthread_create(&thread2, NULL, bar, (void *)"Thread2 is
  ↪  running.");

  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);

  printf("Both threads end.");
  return 0;
}
```

```
$  gcc -o pthreads.x pthreads.c -lpthread
$ ./pthreads.x
Thread1 is running.
Thread2 is running.
Both threads end.
```

Beispiel Callback III [8]

```c
#include <pthread.h>
#include <stdio.h>

void *foo(void *arg) {
  printf("%s\n", (char *)arg);
  return NULL;
}

int main() {
  pthread_t thread1, thread2;
  void *(*bar)(void *) = foo;

  pthread_create(&thread1, NULL, foo, (void *)"Thread1 is
  ↪  running.");
  pthread_create(&thread2, NULL, bar, (void *)"Thread2 is
  ↪  running.");

  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);

  printf("Both threads end.");
  return 0;
}
```

```
$  gcc -o pthreads.x pthreads.c -lpthread
$ ./pthreads.x
Thread1 is running.
Thread2 is running.
Both threads end.
```

Table of Contents

Motivation

Kapselung zusammengehörigen Codes

- Aufteilung der Programmlogik
- Header definiert Schnittstelle (Compile-Zeit)
- Bibliothek liefert Implementation (Link-Zeit und Laufzeit)
- Besserer Überblick
- Vereinfacht testen und debuggen
- Vereinfacht Code-Änderungen
- DRY (*Don't repeat yourself*)
  → Wiederverwendung bestehender (Fremd-)Codes

Einbindung von externem Code

- Präprozessor inkludiert Header
  - Übergabe eines Header-Suchpfades über `-I<pfad>`
- Linker/Loader löst Aufrufe externer Funktionen auf
  - Übergabe eines Library-Suchpfades über `-L<pfad>`
  - Einbindung einer externen Library über `-l<libname>`
- Binary bzw. der Loader muss Pfad zu shared Libs kennen
  - `rpath, runpath`
  - `LD_LIBRARY_PATH`
- Mehr: `http://tldp.org/HOWTO/Program-Library-HOWTO/`

Header

- Inhalt:
  - Funktionsdeklarationen
  - Konstanten/Daten
  - Makros
  - (Funktionsbody)
- Verwendung
  - `#include <foo.h>`
  - `#include "foo.h"`
  - `#include "pfad/foo.h"`
  - Neuer Suchpfad: `-I<pfad>`
- Datei-Endung `.h`

Guard

```c
#ifndef MY_HEADER
#define MY_HEADER

static const int foo = 42;
void bar();

#endif /* MY_HEADER */
```

Anwendungsbeispiel I

main.c

```c
#include <stdio.h>
#include "foo.h"

int main() {
  printf("PI: %d\n", PI);
  helloWorld(3);
  return 0;
}
```

include/foo.h

```c
#ifndef MY_HEADER
#define MY_HEADER
static const int PI = 3;

void helloWorld(int);



#endif /* MY_HEADER */
```

Anwendungsbeispiel I

main.c

```c
#include <stdio.h>
#include "foo.h"

int main() {
  printf("PI: %d\n", PI);
  helloWorld(3);
  return 0;
}
```

include/foo.h

```c
#ifndef MY_HEADER
#define MY_HEADER
static const int PI = 3;

void helloWorld(int);



#endif /* MY_HEADER */
```

Anwendungsbeispiel II

lib/foo.c

```c
#include <stdio.h>
#include "foo.h"

void helloWorld(int eingabe) {
  for (int i = 0; i < eingabe; ++i) {
    printf("Hello World, PI=%d\n", PI);
  }
}
```

lib2/foo.c

```c
#include <stdio.h>
#include "foo.h"

void helloWorld(int eingabe) {
  for (int i = 0; i < eingabe; ++i) {
    printf("HALLO WELT, PI=%d\n", PI);
  }
}
```

Anwendungsbeispiel II

lib/foo.c

```c
#include <stdio.h>
#include "foo.h"

void helloWorld(int eingabe) {
  for (int i = 0; i < eingabe; ++i) {
    printf("Hello World, PI=%d\n", PI);
  }
}
```

lib2/foo.c

```c
#include <stdio.h>
#include "foo.h"

void helloWorld(int eingabe) {
  for (int i = 0; i < eingabe; ++i) {
    printf("HALLO WELT, PI=%d\n", PI);
  }
}
```

Anwendungsbeispiel III

Hierarchie

```
.
|-- main.c
|-- include
|   `-- foo.h
`-- lib
    `-- foo.c
`-- lib2
    `-- foo.c
```

Bau und Ausführung:

```
$ gcc -c main.c -Iinclude
$ gcc -c lib/foo.c
$ gcc -o main.x main.o lib/foo.o

$ ./main.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

Alternative: `gcc -o main.x main.c lib/foo.c`

- Alternative unpraktisch für größere Projekte

Motivation

Bibliothek: Vorbereiteter Container für kompilierten Code und Daten

- Wiederverwendung existierender Fremd-Codes
- Implementation vieler Standard-C-Funktionen in `libc`
  - Standardmäßig eingebunden bei `gcc`
- Implementation mancher Standard-Header in eigene Lib ausgelagert
  - Bsp: `math.h` erfordert `-lm`
  - erfordert explizite Angabe der Lib

  $\rightarrow$ Fehlende Lib resultiert in Linkerfehler
- Arten von Bibliotheken:
  - Static
  - Shared

Static [2]

- Archiv von Objektdateien
- Linker fügt Code in Binary ein
- Vereinfachte Portierung auf anderes System
- File-Endung .a
- Geringfügig schnellere Laufzeit als shared Library

Verwendung

```
$ gcc -c lib/foo.c
$ gcc -c main.c
$ ar rcs libfoo.a foo.o
$ gcc -o main_s.x main.o -L. -lfoo
```

Shared [3]

- Linker: fügt Symbole ein
- Loader: Einbindung bei Programmstart
- Dynamische Einbindung zur Laufzeit möglich [6]
- Kleinere Binary, kürzere Build-Time
- Deutliche Vereinfachung von Code-Updates
- Namenskonvention: *lib*LibraryName.so(.*VERSION*)

Hierarchie

```
.
|-- main.c
|-- include
|   `-- foo.h
`-- lib
    `-- foo.c
`-- lib2
    `-- foo.c
```

Verwendung

```
$ gcc -c -fPIC lib/foo.c -Iinclude -o lib/foo.o
$ gcc -shared -o lib/libfoo.so lib/foo.o
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude
```

Shared [3]

- Linker: fügt Symbole ein
- Loader: Einbindung bei Programmstart
- Dynamische Einbindung zur Laufzeit möglich [6]
- Kleinere Binary, kürzere Build-Time
- Deutliche Vereinfachung von Code-Updates
- Namenskonvention: *lib*LibraryName.so(.*VERSION*)

Hierarchie

```
.
|-- main.c
|-- include
|   `-- foo.h
`-- lib
    `-- foo.c
`-- lib2
    `-- foo.c
```

Verwendung

```
$ gcc -c -fPIC lib/foo.c -Iinclude -o lib/foo.o
$ gcc -shared -o lib/libfoo.so lib/foo.o
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude
```

Shared [3]

- Linker: fügt Symbole ein
- Loader: Einbindung bei Programmstart
- Dynamische Einbindung zur Laufzeit möglich [6]
- Kleinere Binary, kürzere Build-Time
- Deutliche Vereinfachung von Code-Updates
- Namenskonvention: *lib*LibraryName.so(.*VERSION*)

Hierarchie

```
.
|-- main.c
|-- include
|   `-- foo.h
`-- lib
    `-- foo.c
`-- lib2
    `-- foo.c
```

Verwendung

```
$ gcc -c -fPIC lib/foo.c -Iinclude -o lib/foo.o
$ gcc -shared -o lib/libfoo.so lib/foo.o
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude
```

Nützliche Tools

ldd: *Rekursive* Anzeige aller benötigten shared Libraries

nm: Auflistung aller Symbole eines Objekt-Files

objdump: Auslesen von Informationen eines Objekt-Files

readelf: Auslesen von Informationen eines ELF-Objektes

ldconfig: Aktualisieren des System-Caches von Pfaden mit Suchpfaden bzgl. Libraries

Statisch gelinkt

```
$ ldd ./main_s.x
linux-vdso.so.1 (0x00007f481de7f000)
libc.so.6 => /lib64/libc.so.6 (0x00007f481dc56000)
/lib64/ld-linux-x86-64.so.2 (0x00007f481de81000)
```

Dynamisch gelinkt

```
$ ldd ./main_d.x
linux-vdso.so.1 (0x00007f7c9ec91000)
libfoo.so => not found
libc.so.6 => /lib64/libc.so.6 (0x00007f7c9ea68000)
/lib64/ld-linux-x86-64.so.2 (0x00007f7c9ec93000)
$ ./main_d.x
./main_d.x: error while loading shared libraries: libfoo.so: cannot open shared
↪  object file: No such file or directory
```

Was fehlt?

```
$ readelf -d main_d.x | grep NEEDED
0x0000000000000001 (NEEDED)             Shared library: [libfoo.so]
0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
$ ldd ./main_d.x|grep "not found"
libfoo.so => not found
```

Suche nach Library bzgl. einer Abhängigkeit zur Laufzeit(!):

1. `rpath`
2. `LD_LIBRARY_PATH`
3. `runpath`
4. `/etc/ld.so.conf`
5. `Standard-Systempfade`

```
LD_DEBUG=libs ./main_d.x
     87750:  find library=libfoo.so [0]; searching
     87750:   search path=/home/jannek/lib/glibc-hwcaps/x86-64-v3:/home/jannek/lib/glibc-hwcaps/x86-64-v2:/ho
     ↪  me/jannek/lib:/home/jannek/.local/lib/glibc-hwcaps/x86-64-v3:/home/jannek/.local/lib/glibc-hwcaps/x8
     ↪  6-64-v2:/home/jannek/.local/lib:glibc-hwcaps/x86-64-v3:glibc-hwcaps/x86-64-v2:    (LD_LIBRARY_PATH)
     87750:    trying file=/home/jannek/lib/glibc-hwcaps/x86-64-v3/libfoo.so
     87750:    trying file=/home/jannek/lib/glibc-hwcaps/x86-64-v2/libfoo.so
     87750:    trying file=/home/jannek/lib/libfoo.so
     87750:    trying file=/home/jannek/.local/lib/glibc-hwcaps/x86-64-v3/libfoo.so
     87750:    trying file=/home/jannek/.local/lib/glibc-hwcaps/x86-64-v2/libfoo.so
     87750:    trying file=/home/jannek/.local/lib/libfoo.so
     87750:    trying file=glibc-hwcaps/x86-64-v3/libfoo.so
     87750:    trying file=glibc-hwcaps/x86-64-v2/libfoo.so
     87750:    trying file=libfoo.so
     87750:   search cache=/etc/ld.so.cache
     87750:   search path=/lib64/glibc-hwcaps/x86-64-v3:/lib64/glibc-hwcaps/x86-64-v2:/lib64:/usr/lib64/glibc
     ↪  -hwcaps/x86-64-v3:/usr/lib64/glibc-hwcaps/x86-64-v2:/usr/lib64    (system search path)
     87750:    trying file=/lib64/glibc-hwcaps/x86-64-v3/libfoo.so
     87750:    trying file=/lib64/glibc-hwcaps/x86-64-v2/libfoo.so
     87750:    trying file=/lib64/libfoo.so
     87750:    trying file=/usr/lib64/glibc-hwcaps/x86-64-v3/libfoo.so
     87750:    trying file=/usr/lib64/glibc-hwcaps/x86-64-v2/libfoo.so
     87750:    trying file=/usr/lib64/libfoo.so
     87750:
./main_d.x: error while loading shared libraries: libfoo.so: cannot open shared object file: No such file or
↪  directory
```

Userdefinierte Suchpfade

- `rpath`:
  - Optionale Einträge in ELF-Sektion `.dynamic`
  - Linker schreibt ELF-Einträge
- `LD_LIBRARY_PATH`:
  - `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<neuer Suchpfad>`
  - `LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<neuer Suchpfad> ./app`
  - Explizite Angabe einer Bibliothek: `LD_PRELOAD=<library.so>`
- `runpath`:
  - Auswertung von `runpath` nach `LD_LIBRARY_PATH`
    → Überschreibung potentieller Treffer in `runpath` durch User möglich
  - Inkonsistentes Verhalten bei verschiedenen Distributionen möglich (vgl. [7])

RPath (relativ zu aktuellem Verzeichnis)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [lib]
```

RPath (relativ zu Binary)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,\$ORIGIN/lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [$ORIGIN/lib]
```

```
$ ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_LIBRARY_PATH=lib2 ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_PRELOAD=lib2/libfoo.so ./main_d.x
PI: 3
HALLO WELT, PI=3
HALLO WELT, PI=3
HALLO WELT, PI=3
```

RPath (relativ zu aktuellem Verzeichnis)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [lib]
```

RPath (relativ zu Binary)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,\$ORIGIN/lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [$ORIGIN/lib]
```

```
$ ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_LIBRARY_PATH=lib2 ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_PRELOAD=lib2/libfoo.so ./main_d.x
PI: 3
HALLO WELT, PI=3
HALLO WELT, PI=3
HALLO WELT, PI=3
```

RPath (relativ zu aktuellem Verzeichnis)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [lib]
```

RPath (relativ zu Binary)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,\$ORIGIN/lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [$ORIGIN/lib]
```

```
$ ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_LIBRARY_PATH=lib2 ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_PRELOAD=lib2/libfoo.so ./main_d.x
PI: 3
HALLO WELT, PI=3
HALLO WELT, PI=3
HALLO WELT, PI=3
```

RPath (relativ zu aktuellem Verzeichnis)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [lib]
```

RPath (relativ zu Binary)

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,\$ORIGIN/lib,--disable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000000f (RPATH)              Library rpath: [$ORIGIN/lib]
```

```
$ ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_LIBRARY_PATH=lib2 ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_PRELOAD=lib2/libfoo.so ./main_d.x
PI: 3
HALLO WELT, PI=3
HALLO WELT, PI=3
HALLO WELT, PI=3
```

Runpath setzen

Runpath

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,\$ORIGIN/lib,--enable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000001d (RUNPATH)            Library runpath: [$ORIGIN/lib]
```

```
$ ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_LIBRARY_PATH=lib2 ./main_d.x
PI: 3
HALLO WELT, PI=3
HALLO WELT, PI=3
HALLO WELT, PI=3
```

Runpath setzen

Runpath

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,\$ORIGIN/lib,--enable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000001d (RUNPATH)            Library runpath: [$ORIGIN/lib]
```

```
$ ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_LIBRARY_PATH=lib2 ./main_d.x
PI: 3
HALLO WELT, PI=3
HALLO WELT, PI=3
HALLO WELT, PI=3
```

Runpath setzen

Runpath

```
$ gcc -o main_d.x main.c -lfoo -Llib -Iinclude -Wl,-rpath,\$ORIGIN/lib,--enable-new-dtags
$ readelf -d ./main_d.x|grep PATH
0x000000000000001d (RUNPATH)            Library runpath: [$ORIGIN/lib]
```

```
$ ./main_d.x
PI: 3
Hello World, PI=3
Hello World, PI=3
Hello World, PI=3
```

```
$ LD_LIBRARY_PATH=lib2 ./main_d.x
PI: 3
HALLO WELT, PI=3
HALLO WELT, PI=3
HALLO WELT, PI=3
```

Standard-Pfade (Fedora)

Header

```
/usr/lib/gcc/x86_64-redhat-linux/15/include
/usr/local/include
/usr/include
```

Libraries

```
/usr/lib/gcc/x86_64-redhat-linux/15/
/usr/lib64
/usr/lib
/lib
/lib64
```

References

[1] Function Pointer in C
https://www.geeksforgeeks.org/c/function-pointer-in-c/

[2] Static Libraries http://tldp.org/HOWTO/Program-Library-HOWTO/static-libraries.html

[3] Shared Libraries http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

[4] Programming languages — C [Working Draft] http://www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf

[5] C Standard Library header files
https://en.cppreference.com/w/c/header

References ...

[6] Shared Libraries http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html

[7] RPATH und RUNPATH http://blog.qt.io/blog/2011/10/28/rpath-and-runpath/

[8] Thread Management Functions in C https://www.geeksforgeeks.org/c/thread-functions-in-c-c/