



Technische
Universität
Braunschweig

Institut für
Flugführung



Bibliotheken

Erstellung und Verwendung von Bibliotheken

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, 13. Juni 2017

Agenda

- 04. April Kick-Off
- 11. April Projektmanagement
- 18. April Prozessmodelle
- 25. April Versionsverwaltung
- 02. Mai Einführung Arduino/Funduino
- 09. Mai Entwicklungsumgebungen und Debugging
- 16. Mai Dokumentation und Testing
- 23. Mai Dateieingabe und -ausgabe
- 30. Mai GUI-Erstellung mit Qt
- 06. Juni Exkursionswoche
- 13. Juni Bibliotheken**
- 20. Juni Netzwerke
- 27. Juni Projektarbeit
- 04. Juli Projektarbeit
- 11. Juli Vorbereitung der Abgabe

Teil I

Wiederholung

Testing

Anforderungen für gültige Tests

- Testroutinen müssen programmiert werden
→ Kein manuelles Testen
- Anhand der Testkriterien soll überprüft werden, ob die Anforderungen erfüllt wurden oder eine programmierte Funktion das Richtige macht
- Folgende Einträge sollen von der Funktion ausgegeben werden:
 - Name des Tests
 - Autor des Tests
 - Was wird getestet?
 - Dateiname
 - Ergebnis des Tests (erfolgreich/fehlgeschlagen)
- Die Vorlage muss nicht verwendet werden

Wichtige Klassen in Qt

QObject Basisklasse aller Qt-Objekte

QWidget Basisklasse aller UI-Objekte (Benutzerschnittstelle) in Qt.
Kann auch als Fenster genutzt werden.

QString Klasse mit einer Zeichenkette und zahlreichen
Umwandlungsfunktionen

QList Sortierte Liste

QByteArray Array von Bytes. Wird u.a. für das Speichern von Rohdaten
verwendet.

Teil II

Bibliotheken

Bibliothek



- Sammlung von Inhalten und Informationen
- Inhalte werden zur Verfügung gestellt
- Mehrere Mitglieder haben Zugriff auf die Inhalte

Was ist eine Programmbibliothek?

- Sammlung von Methoden, Objekten, Variablen, ...
- Bibliotheken können von Programmen und weiteren Bibliotheken verwendet werden

Bibliotheken sind sinnvoll für...

- ... eine Wiederverwendung von Softwarebauteilen
- ... die Modularisierung von Software
- ... die Erstellung von Schnittstellen
- ... die Wahrung von geistigem Eigentum im Quelltext

Lieferumfang einer Programmbibliothek

1. Schnittstellendefinition

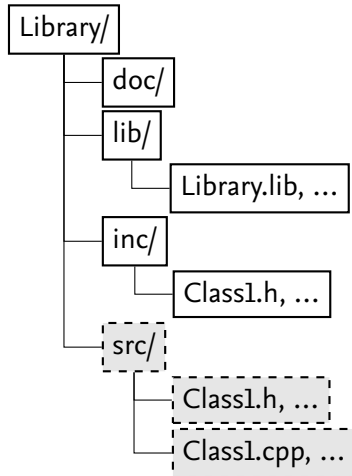
- Header-Dateien
- Deklaration von verfügbaren
 - Variablen
 - Funktionen
 - Klassen
 - ...
- Liegen meistens im `inc` oder `include` Ordner der Bibliothek

2. Gebaute Bibliothek

- Fasst die Objektdateien zusammen
- Liegt meistens im `lib` Ordner der Bibliothek

Programmbibliothek

gängige Ordnerstruktur



C++ Bibliotheken

Dynamische Bibliotheken

.dll	Dynamic Link Library	(Windows)
.so	Shared Object	(Unix)
.dylib	Dynamic Library	(Mac)

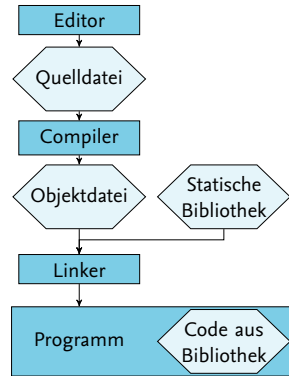
Statische Bibliotheken

.lib	Library	(Windows)
.a	Archive	(Unix, Windows)



Statische Bibliothek

- Inhalte der statischen Bibliothek werden beim Linken in das Programm eingebettet
- Zur Laufzeit wird die Bibliothek nicht benötigt



Statische Bibliothek

Buchbeispiel

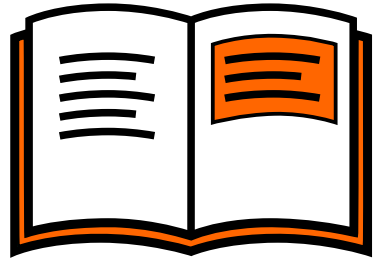


Abbildung 1: Links: Programm, rechts: statische Bibliothek

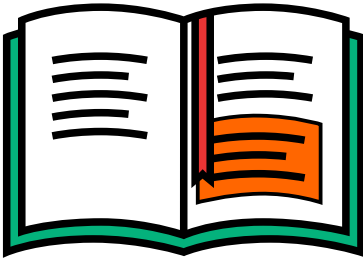


Abbildung 1: Links: Programm, rechts: statische Bibliothek

Vorteile

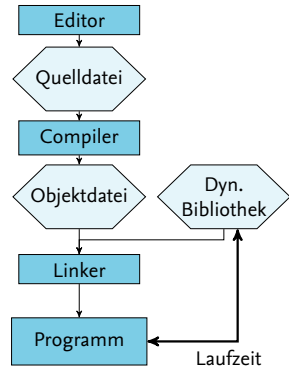
- Geringere Fehleranfälligkeit, da die Bibliothek fester Bestandteil des Programms ist
- Schnellerer Programmstart

Nachteile

- Updates der Bibliothek erfordern ein erneutes Bauen des Programms
- Geringere Speichereffizienz

Dynamische Bibliothek

- Module werden zur Laufzeit geladen
- Bibliothek wird nicht eingebettet
- Mehrere Programme können sich eine Bibliothek teilen
- Austausch der Bibliothek möglich



Dynamische Bibliothek

Buchbeispiel

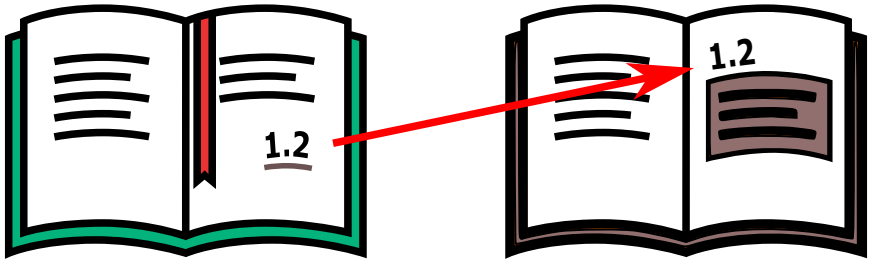


Abbildung 2: Links: Programm, rechts: dynamische Bibliothek

Dynamische Bibliothek

Buchbeispiel

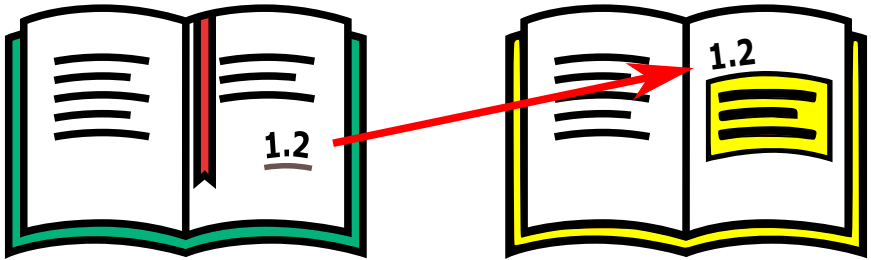


Abbildung 2: Links: Programm, rechts: dynamische Bibliothek

Dynamische Bibliothek

Vor- und Nachteile

Vorteile

- Bibliothek wird einmal in den Speicher geladen
- Speichereffizient
- Zugriff von mehreren Programmen gleichzeitig möglich
 - Bibliothek kann aktualisiert werden, ohne dass das Programm neu gebaut werden muss

Nachteile

- Gefahr, dass eine falsche Bibliothek geladen wird
- Geringfügig längere Ladezeit

Kompatibilität

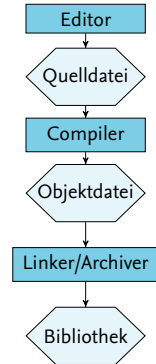
Bei der Einbindung von Bibliotheken müssen die folgenden Faktoren kompatibel sein:

- Plattform
 - Prozessor
 - Betriebssystem
 - ...
- Compiler

GCC, MinGW, Microsoft Visual C/C++, ...
- Debug-Symbole
- Abhängige Bibliotheken

Erstellung einer Bibliothek

1. Quelltext erstellen (Quelldatei)
2. Umwandlung in Maschinencode:
Kompilieren (Objektdatei)
3. Maschinencode und Bibliotheken
zusammenfassen: Linken (Lib)



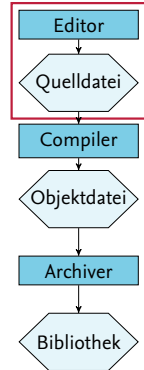
Erstellung einer statischen Bibliothek

Listing 1: code/StaticLib/src/staticLib.h

```
1 void staticLibExample();
```

Listing 2: code/StaticLib/src/staticLib.cpp

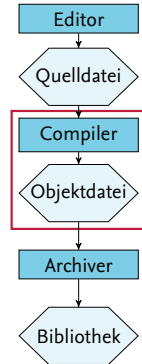
```
1 void staticLibExample(){
2     cout << endl << "Die Bibliothek wurde
   erfolgreich geladen" << endl;
3 }
```



Erstellung einer statischen Bibliothek

Bauen der Objektdaten:

```
$ g++ -c staticLib.cpp
```



Erstellung einer statischen Bibliothek

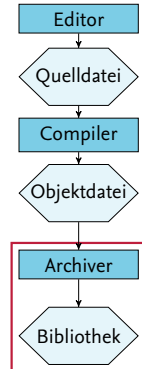
Archivieren der Objektdateien:

```
$ ar rs libstaticLib.a staticLib.o
```

Wichtig

Bibliotheken müssen je nach Compiler nach einem definierten Muster benannt werden.

MinGW: `lib<Name der Bib>.a`

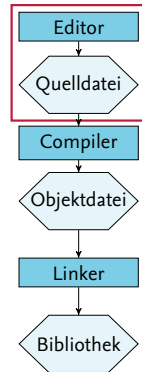


Erstellung einer dynamischen Bibliothek

Unter Windows:

`__declspec(dllexport)` fügt Funktionen oder Klassen zur dll-Bibliothek hinzu

`__declspec(dllimport)` wird bei der Nutzung von Funktionen oder Klassen aus einer dll-Bibliothek benötigt



Erstellung einer dynamischen Bibliothek

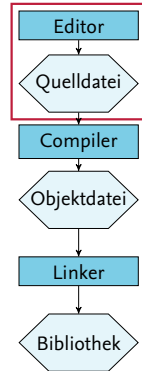
Aus praktischen Gründen werden Makros definiert:

Listing 3: code/DynamicLib/src/lib_global.h

```

1  #ifdef __WIN32__
2  #ifdef BUILD_LIB
3  #define LIB_EXPORT __declspec(dllexport)
4  #else
5  #define LIB_EXPORT __declspec(dllimport)
6  #endif
7  #else
8  #define LIB_EXPORT // Leer bei Unix
9  #endif

```



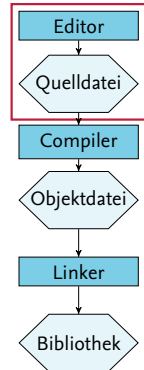
Erstellung einer dynamischen Bibliothek

Listing 4: code/DynamicLib/src/dynLib.h

```
1 #include "lib_global.h"
2
3 LIB_EXPORT void dynLibExample();
```

Listing 5: code/DynamicLib/src/dynLib.cpp

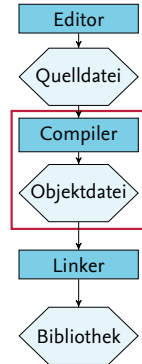
```
1 void dynLibExample(){
2     cout << endl << "Die Bibliothek wurde
3     erfolgreich geladen" << endl;
4 }
```



Erstellung einer dynamischen Bibliothek

Bauen der Objektdaten:

```
$ g++ -c -DBUILD_LIB dynLib.cpp
```



Erstellung einer dynamischen Bibliothek

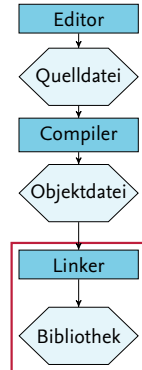
Erstellung der dynamischen Bibliothek:

```
$ g++ -shared -o dynLib.dll dynLib.o
```

ODER

Erstellung der dynamischen Bibliothek
gemeinsam mit der Import-Bibliothek:

```
$ g++ -shared -o dynLib.dll dynLib.o -Wl  
,--out-implib,libdynLib.a
```



Verwendung einer Bibliothek

Compiler und Linker benötigen die folgenden Informationen:

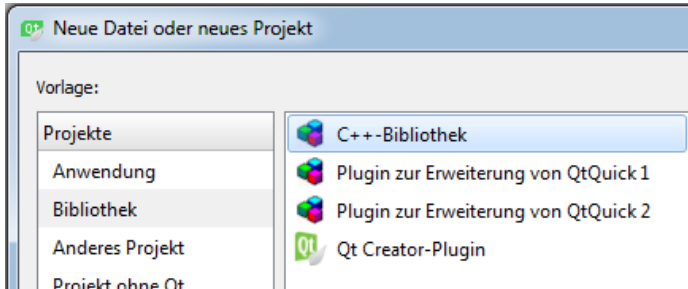
- Verzeichnis der Header-Dateien -I<Ordner>
I steht für include
- Verzeichnis der Bibliothek -L<Ordner>
- Name der Bibliothek -l<Name>

Diese Informationen werden über Optionen an Compiler und Linker übergeben.

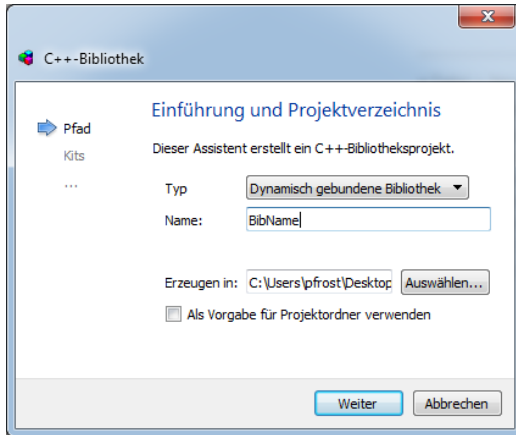
```
$ g++ -o test.exe main.cpp -L. -ldynLib
```

Ein Punkt stellt den aktuellen Ordner dar.


Erstellung einer Bibliothek in Qt




Erstellung einer Bibliothek in Qt



Einbinden einer Bibliothek in Qt

 Bibliothek hinzufügen

Typ

 Details

Zusammenfassung

Externe Bibliothek

Geben Sie den Pfad und die Include-Pfade der Bibliothek an

Bibliotheksdatei:

Include-Pfad:

Plattform Linken: dynamisch

☒ Linux ☒ dynamisch ☐ statisch

☒ Mac

Mac: Bibliothek

☒ Windows ☒ Bibliothek ☐ Framework

Windows:

☒ Bibliothek innerhalb "debug" oder "release" Unterordner

☐ Suffix "d" für Debug-Version anfügen

☐ Suffix "d" für Release-Version entfernen

Einbinden einer Bibliothek in Qt

Anschließend wird die folgenden Zeilen automatisch in die Projektdatei hinzugefügt und die Bibliothek kann verwendet werden.

Listing 6: code/Qt/QtProjekt.pro

```
1 win32:CONFIG(release, debug|release): LIBS += -LPWD/../../
    APILiveDemoBib/builds/Desktop_Qt_5_8_0_MinGW_32bit-Debug/release
    / -lAPILiveDemoBib
2 else:win32:CONFIG(debug, debug|release): LIBS += -LPWD/../../
    APILiveDemoBib/builds/Desktop_Qt_5_8_0_MinGW_32bit-Debug/debug/
    -lAPILiveDemoBib
3
4 INCLUDEPATH += PWD/../../APILiveDemoBib
5 DEPENDPATH += PWD/../../APILiveDemoBib
```

Praxisdemonstration Erstellung einer Bibliothek in QT

Teil III

Projektarbeit

Anmeldung Projektmappe per E-Mail

Listing 7: Betreff der E-Mail

```
1 Anmeldung API-Projektmappe
```

Listing 8: Inhalt der E-Mail

```
1 Vorname:  
2 Nachname:  
3 Matrikelnummer:  
4 GitHub-Benutzer:  
5 GitHub-Team:  
6 GitHub-Repository-URL: .git  
7 GitHub-Wiki-URL: .wiki.git
```

An die folgende E-Mail-Adresse:
api-iff@tu-braunschweig.de

Anmeldung Projektarbeit per E-Mail

Anmeldung

- Jeder muss sich bis zum 30.06.2017 anmelden.
- Für die E-Mail sollen die Inhalte von Listing 7 und Listing 8 verwendet werden.
- Oder direkt den Link verwenden: api-iff@tu-braunschweig.de

2. Zyklus vom API-Spiralmodell abschließen

Aufgabe 1

Schließen Sie den zweiten Zyklus des API-Spiralmodells ab.

Externer-Code Review

Ankündigung

Am 27. Juni besteht die Möglichkeit einen externen Code-Review durchzuführen. Diesbezüglich werden die anwesenden Gruppen vermittelt.

Fragen?

Gibt es noch Fragen?