Proseminar "C-Programmierung – Grundlagen und Konzepte"

Ausarbeitung von

Ludwig Eisenblätter

zum Thema

"Modulare Programmierung und Bibliotheken"

Inhaltsverzeichnis

1. Motivation und Einleitung

Bei der Entwicklung von Software entsteht, je nach Umfang des geplanten Projekts, schnell eine unüberschaubar große Menge von Code. Jeder Programmierer wird demnach bestrebt sein ein komplexes bzw. umfangreiches Softwareprojekt in einzelne Bereiche oder Strukturen aufzuteilen.

In der Regel geschieht dies indem man versucht einzelne logische Teilblöcke eines Projekts zu identifizieren und als Einheit aufzufassen. Diese Einheiten können dann für sich entwickelt werden und bieten eine vorher spezifizierte Leistung an. Die Aufteilung von Software in einzelne logische Teilblöcke nennt man Modularisierung.

Durch diese Aufteilung wird auch die Wiederverwendung von Code ermöglicht, da bereits existierende Softwarelösungen einfach, oder nur mit geringen Änderungen, in neue Systeme übernommen werden können. Auch die separate Lizenzierung von Programmteilen (Modulen) spielt vor allem in der kommerziellen Softwareentwicklung eine Rolle.

Die Modularisierung von Software ermöglicht in vielen Programmiersprachen außerdem das separate Übersetzen (kompilieren) von Programmteilen und damit die Bereitstellung von Bibliotheken auf Basis von Modulen.

Die Ausarbeitung zu diesem Proseminar-Thema führt im Kapitel 2 zunächst die Grundprinzipien und Eigenschaften von Modulen sowie deren Umsetzung in der Programmiersprache C ein.

Im 3. Kapitel wird dargestellt wie die Modularisierung von Software zur Erstellung und Verwendung von Bibliotheken führt. Den Abschluss bildet ein Programmierbeispiel in C.

2. Modulare Programmierung

2.1. Allgemeines

Im Zusammenhang mit Modularisierung spricht man auch von einer Aufteilung in eine "obere und untere Abstraktionsebene".[1]

Die obere Abstraktionsebene (auch: Programmierung im Großen) hat dabei das Ziel die Struktur einer Software zu erfassen und von der konkreten Implementierung zu abstrahieren. Auf dieser Ebene werden Teilfunktionalitäten und logische Einheiten innerhalb der Software bestimmt und in Module aufgeteilt.

Dabei wird entschieden wie die einzelnen Module gestaltet werden (z.B. die Schnittstelle) und wie diese angeordnet werden. Außerdem wird betrachtet wie sich einzelne Programmeinheiten zusammensetzen und verhalten sollen. Schließlich wird auch untersucht, inwiefern Module miteinander agieren um eine Funktionalität bereitzustellen und wo gegebenenfalls Abhängigkeiten zwischen Modulen bestehen.

1 Vgl. Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, S. 147

Die untere Abstraktionsebene (auch: Programmierung im Kleinen) hingegen betrachtet die konkrete Umsetzung der Module. Dabei geht es im Wesentlichen um Steueranweisungen (Ablauf- und Kontrollstrukturen) sowie um Funktionale Abstraktion (Bereitstellung von festgelegten Schnittstellen) und Datenabstraktion (Datenstrukturen, Zugriff auf Daten).

2.2. Definition und Eigenschaften von Modulen

Eine leicht verständliche und zutreffende Definition eines Moduls liefert Wikipedia:

„ Ein Modul ist eine abgeschlossene funktionale Einheit einer Software, bestehend aus einer Folge von Verarbeitungsschritten und Datenstrukturen.

Inhalt eines Moduls ist haufig eine wiederkehrende Berechnung oder Bearbeitung von Daten, die mehrfach durchgefuhrt werden muss.“[2]

Zu den wichtigsten Eingenschaften eines Moduls gehören unter anderen:

- Trennung von Schnittstelle und Implementation

  Ein Modul bietet eine "Kapselung" seiner Daten und Funktionen. Dabei definiert eine festgelegte Schnittstelle nach außen hin die Datenelemente bzw. Funktionen die ein Modul anbietet. Die eigentlichen Daten bzw. der Programmcode ist aber in der Implementation gekapselt.[3]

- Module können geschachtelt werden und erhalten dadurch eine Aufrufhierarchie entsprechend der aufrufenden bzw. aufgerufenen Module.

- Module können wiederverwendet werden, da jedes Modul ein festgelegtes Angebot an Funktionen bzw. Daten liefert. Sie tragen damit zur Vermeidung von redundantem Code bei.

- Module helfen große Softwareprojekte zu strukturieren. Durch die "Aufsplittung" eines Projekts in Teilbereiche bzw. Module können auch einzelne Programmierteams besser auf genau spezifizierte Aufgaben (Module) verteilt werden.

2 http://de.wikipedia.org/wiki/Modul_%28Software%29

3 Zum Thema Datenkapselung und Geheimnisprinzip vgl. auch Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, S. 148

2.3. Module in C

2.3.1. Vorwort

Vor der Einführung von Modulen in C soll hier noch einmal kurz grundlegendes zur Deklaration / Definition und dem Geltungsbereich von Variablen und Funktionen gesagt werden.

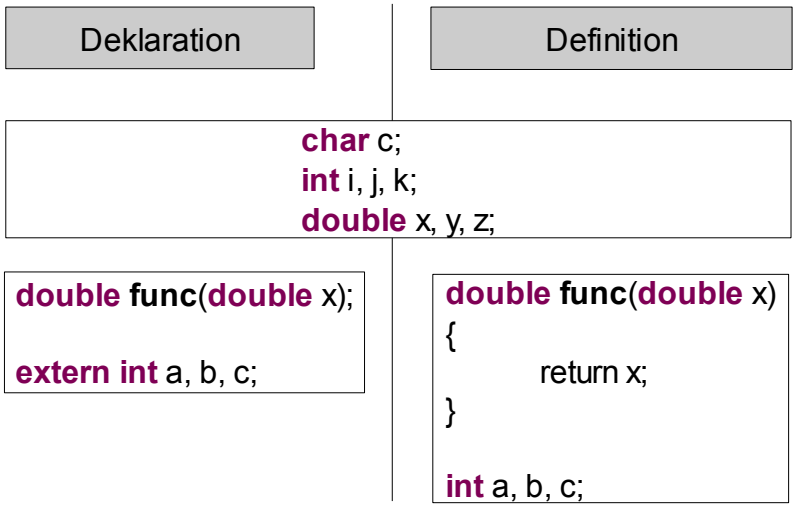

Abb. 1 verdeutlicht die Unterschiede zwischen einer Deklaration und einer Definition in C.

Variablen werden bei ihrer Deklaration immer auch gleich definiert.[4] Dies ist notwendig damit der Übersetzer weiß wie viel Speicherplatz für die Variable reserviert werden muss. Funktionen können deklariert werden (linke Seite) und später im Quelltext, oder sogar in einer anderen Quelldatei, erst definiert werden. Damit kann man Funktionen (bzw. Funktionssignaturen) schon deklarieren und auch im Quelltext benutzen bevor sie endgültig definiert wurden. Gleiches gilt auch für mit „extern“ gekennzeichnete globale Variablen. Dieses Schlüsselwort im Quelltext zeigt an, dass die Variable auch über Dateigrenzen hinweg sichtbar ist. Sie kann sowohl in der deklarierenden Datei oder auch in einer anderen Datei definiert werden.[5]

Geltungsbereich.c

```
int i, j, k;
double x, y, z;

void func();

extern int a, b, c;

void func()
{
      int r = 0;
      Static int s = 0;
}
```

Zur Darstellung des Geltungsbereichs von Variablen dient die nebenstehende C-Datei.

Die Variablen im oberen Teil der Datei sind (im Kontext dieser Datei) globale Variablen, da sie vor allen anderen Deklarationen stehen. Sie sind damit im Quelltext der ganzen Datei verfügbar.

Funktionen die keinen Rumpf besitzen (oberer Teil von func()) sind Funktionsdeklarationen. Da in keinem C-Programm eine Funktion mit gleichem Namen mehrmals vorkommen darf, sind Funktionsdeklarationen (im Kontext des C-Programms) immer global verfügbar.

4 Vgl. http://de.wikibooks.org/wiki/C-Programmierung:_Variablen_und_Konstanten
5 Vgl. Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, S. 124

Variablen innerhalb von Funktionen (in diesem Fall „r“) sind nur während des Funktionsaufrufs deklariert und definiert.

Werden Variablen in Funktionen mit dem Schlüsselwort „static“ deklariert bleiben sie auch nach dem Funktionsaufruf erhalten und sind beim nächsten Aufruf unverändert abrufbar.[6]

2.3.2. Umsetzung von Modulen in C

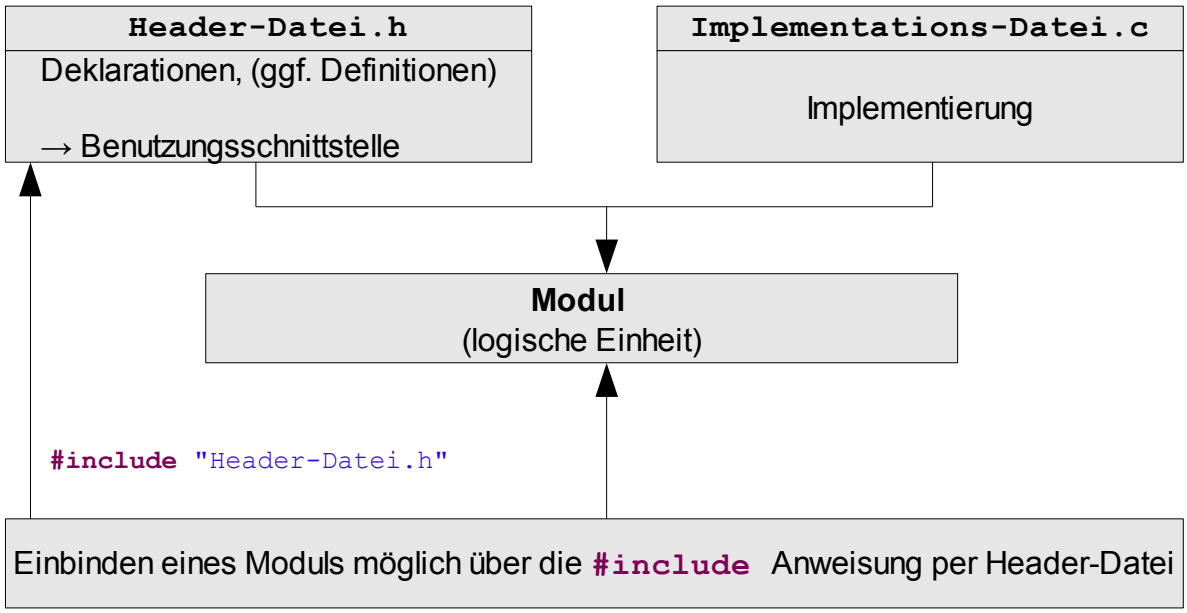

Abbildung 2: Schematische Ansicht der Umsetzung von Modulen in C

In C besteht ein Modul immer aus einer Header-Datei (Dateiendung *.h) und einer Implementations-Datei (Dateiendung *.c). Beide zusammen bilden eine logische Einheit.[7]

Die Header Datei enthält alle für die Benutzung notwendigen Deklarationen des Moduls, während der tatsächliche Quellcode für die Ausführung in die Implementationsdatei ausgelagert ist. Die Header-Datei bildet somit die Benutzungsschnittstelle des Moduls.

Um ein Modul in einem C-Programm zu benutzen wird über die Präprozessor-Anweisung `#include` die Header-Datei des Moduls eingebunden.[8] Danach stehen die Funktionen des Moduls zur Verfügung. Um ein solches C-Programm zu übersetzen sind jedoch weitere Schritte nötig die später erläutert werden.

Um eine mehrfache Einbindung eines Moduls per include-Befehl (z.B. durch geschachtelte Modul-Aufrufe) zu verhindern, bietet es sich an sogenannte „bedingte Includes“ zu benutzen. Abbildung 3 verdeutlicht die Vorgehensweise:

6 Vgl. Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, S. 127
7 Vgl. Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, S. 132
8 Vgl. Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, S. 131

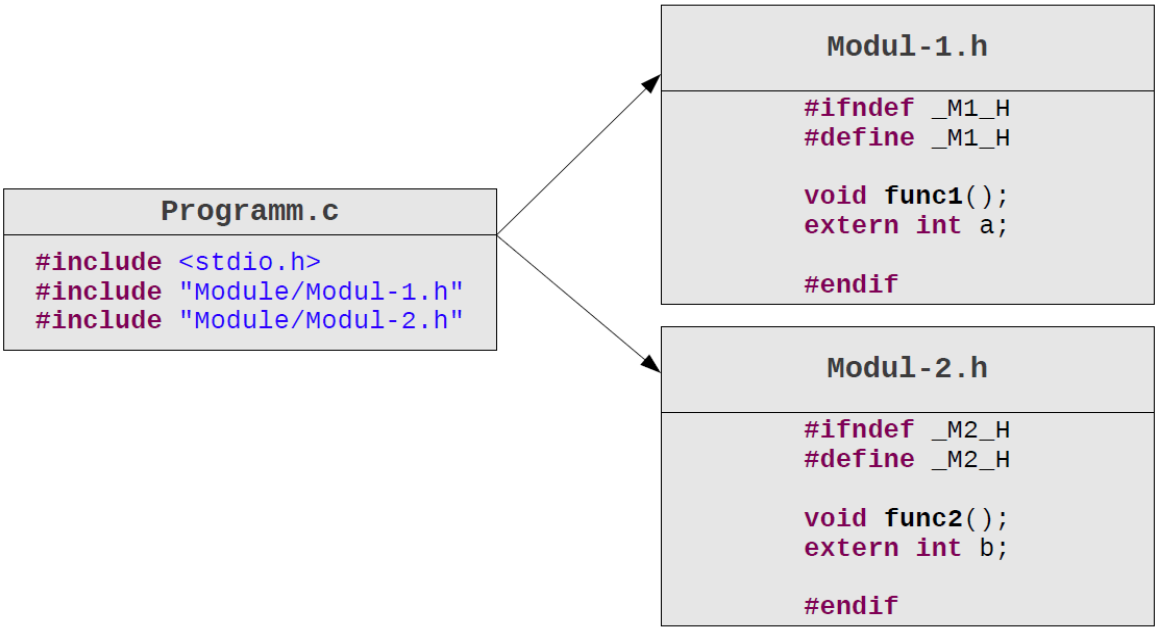

Abb. 3: „Bedingte Includes“ zur Einbindung von Modulen nutzen

Über die Anweisung `#define` wird per Präprozessor-Anweisung eine Konstante für jedes Modul gesetzt (hier „_M1_H“ bzw. „_M2_H“). Bei jedem „include“ Aufruf wird vom Präprozessor geprüft ob das entsprechende Modul bereits eingebunden wurde (Konstante ist gesetzt) oder nicht (Konstante nicht gesetzt). Nur bei der ersten Ausführung von „include“ werden dann die Funktionen und externen Variablen des Moduls (hier „func1()“, „func2()“, a, b) deklariert.[9]

Das obige Beispiel zeigt auch, dass Module nicht zwangsläufig im gleichen Verzeichnis wie die einbindende C-Datei liegen müssen. In diesem Fall gibt es beispielsweise einen Unterordner „Module“ aus dem die beiden Header-Dateien eingebunden werden (siehe „#include“-Anweisungen in „Programm.c“).

2.3.3. Übersetzen von Modulen

Um Module (und Programme die Module enthalten) übersetzen zu können müssen die Module erst separat in eine Objektdatei übersetzt werden. Danach werden sie zu dem (ebenfalls als Objektdatei vorliegenden) Hauptprogramm hinzu „gelinkt“ und eine ausführbare Version des Programms erzeugt.

Um diesen Vorgang besser verstehen zu können ist es hilfreich sich die verschiedenen Zustände einer Quelldatei während des Übersetzungsvorgangs zu verdeutlichen. Abbildung 4 zeigt dies als Übersicht:

9 Beispiel und Erläuterung vgl. Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, S. 135

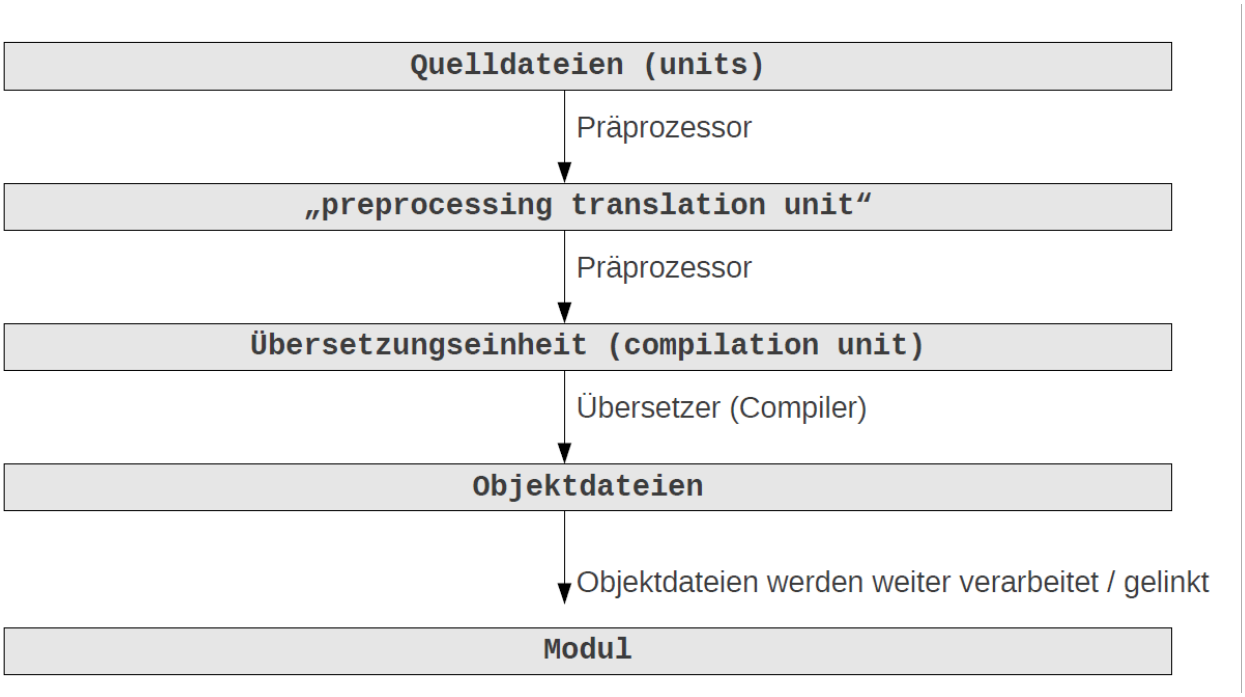

Abb. 4: Übersetzungs- und Link-Vorgang schematisch dargestellt

Eine einzelne Quelldatei (unit) wird vom Präprozessor als sogenannte „preprocessing translation unit“ verarbeitet.

In der resultierenden Übersetzungseinheit (compilation unit) sind dann alls Präprozessor-Anweisungen durchgeführt worden. Insbesondere wurden also auch alle Ersetzungen (z.B. Konstanten) und Erweiterungen (z.B. Einfügen von Funktionen durch „#include“-Anweisungen) berücksichtigt. [10]

Der Übersetzer (Compiler) übersetzt die Quelldatei anschließend in eine Objektdatei (beim Übersetzer „gcc“ wäre dies z.B. der Parameter „-c“).

Die fertigen Objektdateien werden abschließend zu einem fertigen Programm (ausführbare Datei) oder einem Modul (Objekt-Datei) verknüpft. Ein Modul als Objekt-Datei kann wiederum mit weiteren Objektdateien zu einem fertigen Programm oder einem größeren Modul verknüpft werden.

10 Vgl. Zur Beschreibung des Vorgangs auch http://en.wikipedia.org/wiki/Compilation_unit

In der Praxis ergeben sich nun verschiedene Möglichkeiten ein Modul, bzw. ein Programm das Module enthält, zu übersetzen.

- Manuelles Übersetzen und Verknüpfen

  Für das obige Beispiel würde eine Übersetzung mit gcc etwa so aussehen:

  ```
  gcc -c modul-1.c
  gcc -c hauptprogramm.c
  gcc -o Hauptprogramm modul-1.o hauptprogramm.o
  ```

  Mit den beiden ersten Befehlen werden die Objektdateien für das Modul und das Hauptprogramm erzeugt. Anschließend werden diese zu einem fertigen ausführbaren Hauptprogramm verknüpft.

- Verwaltung und Übersetzen durch Makefiles

  Makefiles sind eine weitere recht verbreitete Methode um den Übersetzungsvorgang für viele Dateien überschaubar zu halten. Letztendlich wird dabei in „make-Files“ definiert welche Dateien übersetzt werden sollen und welche Abhängigkeiten zwischen ihnen bestehen. Davon ausgehend wird durch das Programm „make“ der Übersetzungsvorgang durchgeführt.[11]

- Intelligente Systeme zur Verwaltung

  Es existiert eine Vielzahl von Systemen, als Beispiel sei hier „waf“ genannt.[12]

- Automatisches „Build“ (inkl. Abhängigkeitsprüfung) durch IDE's

  Eine weitere Möglichkeit ist die Nutzung einer IDE (Integrated Development Environment), welche für Projekte ein sogenanntes „Build“ durchführen können und damit alle geänderten Quelldateien neu übersetzen und zusammenfügen. Als prominentes Beispiel sei hier Eclipse[13] genannt.

11 Für weitere Informationen siehe auch http://en.wikipedia.org/wiki/Makefile
12 Siehe auch http://code.google.com/p/waf/
13 Die Projekthomepage ist unter http://www.eclipse.org erreichbar.

3. Bibliotheken

3.1. Motivation & Einführung

Bibliotheken sind eine Sammlung von Funktionen, Unterprogrammen, Klassen oder Werten für gewisse Aufgabenbereiche. Sie entstehen aus dem Bedürfnis, bereits vorhandenen Code oder spezielle, getestete und für robust gehaltene Funktionen auch für den späteren Einsatz wieder bereitzustellen. Sie sind in der Regel keine lauffähigen Programme, sondern Hilfsmodule die Programmen zur Verfügung gestellt werden.[14]

Die Nutzung von Bibliotheken in C resultiert in erster Linie daraus, dass der Standard-Funktionsumfang von C nicht besonders groß ist. Schon für einfache String Operationen oder mathematische Berechnungen wird daher auf eine Implementierung der C-Standardbibliothek zurückgegriffen.

Auch selbst geschriebener Code kann zur späteren Wiederverwendung in eine Bibliothek ausgelagert werden und damit auch Dritten zur Verfügung gestellt werden.

Bibliotheken bieten damit außerdem eine Möglichkeit Code zu zentralisieren. Dies ermöglicht eine leichtere Wartbarkeit, da immer nur die Bibliothek an sich aktualisiert wird und nicht alle Quellen die diese Bibliothek nutzen. Auch sicherheitskritische Updates können so leichter eingespielt und überwacht werden.

Seit der Einführung von C hat sich eine große Menge von öffentlich zugänglichen Bibliotheken für fast jedes denkbare Anwendungsgebiet und viele Spezialprobleme entwickelt. Eine kurze Übersicht über einige populäre Bibliotheken gibt die folgende Aufzählung:

Die **glibc (GNU C Library)**[15] ist eine Implementierung der Standard-C-Bibliothek. Sie wird vom GNU-Projekt bereitgestellt und ist für verschiedene Plattformen (u.a. x86, SPARC) verfügbar. Die glibc bietet neben den für die Standard-C-Bibliothek geforderten Funktionen auch zusätzliche (nicht standardisierte) Erweiterungen.

Die Bibliothek **gtk+ (Gimp Toolkit)**[16] stellt Funktionen zur Erstellung grafischer Benutzeroberflächen (GUI, Graphical User Interface) zur Verfügung.

Die **GLib (Gnome Library)**[17] wird von den Entwicklern von *gtk+* bereitgestellt und ist aus dem *„gtk+“*-Projekt entstanden. Viele nützliche Funktionen die für dieses Projekt entwickelt wurden, aber nicht direkt mit GUI-Programmierung zu tun haben wurden in die GLib ausgelagert. Dadurch können Programmierer diese nutzen ohne den „Ballast“ des GUI-Teils mit zu übernehmen. Die GLib bietet eine Vielzahl von Funktionen für die verschiedensten Bereiche, u.a. objektorientierte Programmierung, komplexe Datenstrukturen (wie balancierte Binärbäume und Listen), Funktionen für Threads, Zeitfunktionen, Zeichenkettenfunktionen und Funktionen für Speicherzugriffe.[18]

Die **NAG C-Library**[19] ist eine Bibliothek für numerische Funktionen. Sie bietet u.a. Funktio-

14 http://de.wikipedia.org/wiki/Bibliothek_%28Programmierung%29
15 Die Projekthomepage ist unter http://www.gnu.org/software/libc/ erreichbar.
16 Die Projekthomepage ist unter http://gtk.org/ zu erreichen.
17 Siehe dazu auch den Link für gtk+.
18 Siehe auch: http://de.wikipedia.org/wiki/GLib
19 Die Projekthomepage ist unter http://www.nag.co.uk/ zu erreichen.

nen in den Bereichen Numerische Integration, Interpolation, Zufallszahlen und nichtlineare Gleichungen.

Die **Libwww-Bibliothek**[20] ist eine Web-API, über die verschiedene Funktionen für Internet-basierte Dienste bereitgestellt werden. Unterstützt werden u.a. die Protokolle FTP, HTTP, Telnet und file sowie SSL-Verschlüsselung.

3.2. Dynamische und statische Bibliotheken

Wie oben beschrieben stellen Bibliotheken Funktionen und Daten für andere Module bzw. Programme zur Verfügung. Im Übersetzungsprozess werden diese Funktionen bzw. Daten über Verknüpfungen („links“) mit dem einbindenden Programm verknüpft. Dieser Vorgang wird von dem sogenannten Linker durchgeführt.[21]

Besteht ein Programm aus mehreren Objektdateien (z.B. einer Bibliothek und einem Hauptprogramm) verweisen die Objektdateien über Symbole (relative Speicheradressen bzw. Verweise) aufeinander. Beim Link-Vorgang löst der Linker diese Symbole auf und ersetzt die Speicherverweise durch absolute oder eindeutig zuweisbare Adressen.

Generell unterscheidet man dabei statisches und dynamisches Linken.[22] Beim statischen Linken werden die Objektdateien der Bibliothek(-en) und Module bzw. Programme vom Übersetzer zusammengeführt und anschließend wird vom Linker eine ausführbare Datei erzeugt indem er alle symbolischen Verweise („links“) innerhalb der verschiedenen Objekte auflöst. Statische Bibliotheken haben unter UNIX „.a“ und unter Windows „.lib“ als Dateiendung.

Beim dynamischen Linken werden alle Funktionen der Bibliothek erst beim Laden (load time) oder Ausführen (execution time) des fertigen Programms geladen. Der Linker erstellt beim Übersetzungsvorgang lediglich eine Übersicht über die benutzten Bibliotheken und den Index der genutzten Funktionen. Unter Windows wird für dynamische Bibliotheken die Dateiendung „.dll“ und unter UNIX „.dso“ (dynamic shared object) verwendet.

3.3. Benutzen von Bibliotheken in C

Bibliotheken werden, wie Module auch, über eine `#include` Anweisung im Kopf der C-Quelldatei eingebunden. Allerdings werden Bibliotheken immer in spitze Klammern gesetzt (z.B. „`#include <stdio>`“). Der Übersetzer erkennt daran dass es sich um eine Bibliothek handelt. Der Übersetzer sucht Bibliotheken immer in dem ihm mitgeteilten Standardpfad. Wenn eigene Bibliotheken oder Bibliotheken in anderen Ordnern eingebunden werden sollen muss dies dem Übersetzer per Parameter mitgeteilt werden. Im folgenden Kapitel werden diese Parameter an einem einfachen Beispiel erläutert.

20 Die Projekthomepage ist unter http://www.w3.org/Library/ zu erreichen.
21 s.a. http://de.wikipedia.org/wiki/Linker_%28Computerprogramm%29
22 s.a. http://en.wikipedia.org/wiki/Library_%28computing%29#Static_libraries

3.4. Anwendungsbeispiel

3.4.1. Allgemeines

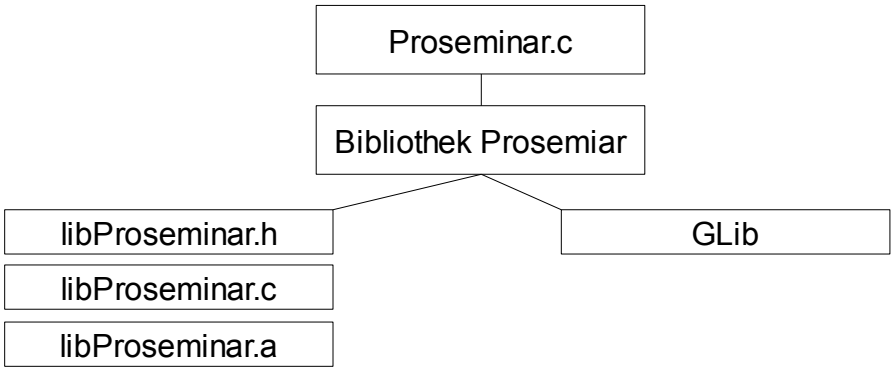

Abb. 5: Aufbau des Anwendungsbeispiels.

Das Programm „Proseminar.c“ benutzt die Funktionen der Bibliothek „Proseminar“. Diese Bibliothek besteht aus verschiedenen Quelldateien und benutzt ihrerseits wiederum eine Funktion der GNU Library (GLib).

Die Bibliothek Proseminar stellt für Anschauungszwecke nur eine Funktion „demo()“ bereit.

```
#ifndef SEMINARBIBLIOTHEK_H
#define SEMINARBIBLIOTHEK_H

void demo();

#endif
```

Diese wird in der Hader-Datei „libProseminar.h“ deklariert.

Die Implementation in „libProseminar.c“ ist entsprechend simpel:

```
#include <glib.h>
#include <glib/gprintf.h>

void demo()
{
g_printf("Benutzen der GLib Funktion g_printf()\n");
}
```

Die Implementation benutzt die GLib-Funktion „gprintf()“, welche im Wesentlichen wie die normale „printf()“-Funktion arbeitet. Beim Aufruf von „demo()“ wird der entsprechende String auf der Konsole ausgegeben.

Um nun die Bibliothek und das Programm übersetzen zu können müssen dem Übersetzer noch die jeweiligen Parameter übergeben werden. Ein einfaches Übersetzen mit gcc funktioniert nicht, wie die entsprechende Ausgabe auf der Konsole zeigt:

```
gcc -o libProseminar.o libProseminar.c

libProseminar.c:1:18: fatal error: glib.h: Datei oder Verzeichnis nicht gefunden

compilation terminated.
```

Dies liegt daran dass dem Übersetzer der sogenannte „Include-Path“, also der Pfad in dem die im Quelltext benutzten Header-Dateien liegen, und eine Angabe über die Bibliothek mitgeteilt werden muss. Bei gcc wird dies über die Parameter „`-I`“ (für Include) und „`-l`“ (für library) gemacht.

Die Werte für die Parameter können entweder manuell „herausgesucht“ werden oder automatisch mit dem populären Tool „pkg-config“ erledigt werden. Das folgende Kapitel gibt eine kurze Einführung in dieses Werkzeug.

3.4.2. Exkurs „pkg-config“

Das Programm „pkg-config“[23] (für „Package-Configuration“) steht neben Unix-Systemen auch für andere Plattformen zur Verfügung (z.B. Windows) und ist ein Werkzeug zum Bereitstellen von Compiler- und Linker-Parametern.

Pkg-config bezieht seine Daten aus sogenannten „Metafiles“ (diese heißen entsprechend der Bibliothek für die sie geschrieben werden und haben die Endung „.pc“). Die Metafiles enthalten u.a. Name, Beschreibung, Version, Abhängigkeiten, bekannte Konflikte mit anderen Paketen, Verzeichnisangaben für die Include-Anweisungen und nicht zuletzt die benötigten Compiler-Parameter für das entsprechende Paket.

Die Metafiles werden manuell oder bei der Installation einer Programmbibliothek in einem von der Umgebungsvariable „PKG_CONFIG_PATH“ festgelegten Ordner abgelegt. Für weitere Informationen hierzu und den unten folgenden Befehlen ist das Manual[24] nützlich.

Die gängigsten Befehle für pkg-config sind:

`pkg-config --help`	Zeigt eine Liste aller Befehle inklusive kurzen Kommentaren an.
`pkg-config --list-all`	Zeigt alle Bibliotheken bzw. Module an für die pkg-config die entsprechenden Parameter bereitstellen kann.
`pkg-config --libs`	Gibt für die gewählte Bibliothek die benötigten Präprozessor- und Compiler-Flags aus. Auch alle evtl. bestehenden Abhängigkeiten der einzelnen Bibliotheken werden berücksichtigt.
`pkg-config --cflags`	Gibt für die gewählte Bibliothek die „Compiler-Flags“, z.B. Angaben zum „Link“-Vorgang.

Ein typischer Aufruf von `pkg-config` auf einem Linux-System könnte z.B. so aussehen:

```
pkg-config --cflags --libs glib-2.0
```

23 Die Projekthomepage ist unter http://www.freedesktop.org/wiki/Software/pkg-config erreichbar.
24 Eine ausführlichere Dokumentation ist auf http://linux.die.net/man/1/pkg-config zu finden.

3.4.3. Übersetzen des Anwendungsbeispiels

Die Bibliothek aus dem o.g. Beispiel kann nun übersetzt werden (alle folgenden Befehle werden auf einer Linux-Kommandozeile ausgeführt):

```
gcc -c -o libProseminar.o libProseminar.c `pkg-config --cflags glib-2.0`
```

Die Linux-Kommandozeile fügt dabei die Ausgabe des in einfachen Anführungszeichen stehenden Befehls von pkg-config automatisch ein.

Aufgeschlüsselt würde sich als vollständiger Aufruf somit dieser Befehl ergeben:

```
gcc -c -o libProseminar.o libProseminar.c -I/usr/include/glib-2.0
-I/usr/lib/i386-linux-gnu/glib-2.0/include
```

Damit wurde die Bibliothek nun übersetzt und liegt als Objektdatei vor. Sie könnte nun direkt gegen das Programm gelinkt werden in dem sie benutzt wird (sofern das Programm auch als Objekt-Datei vorliegt).

Im Abschnitt 3.2. wurde auf den Unterschied zwischen statischen und dynamischen Bibliotheken eingegangen.

Zur Verdeutlichung wird hier noch gezeigt wie die Objektdatei in ein vor-kompiliertes Archiv (statische Bibliothek) gepackt werden kann. Dazu wird das Programm „ar“[25] unter Linux verwendet:

```
ar -rcs libProseminar.a libProseminar.o
```

Hierbei können auch mehrere Objektdateien zu einem Archiv hinzugefügt werden.

Zu beachten ist, dass dies eine statische Unix-Bibliothek mit der Endung „.a“ erzeugt. Unter Windows müssten andere Programme zum Packen benutzt werden. Dort haben statische Bibliotheken außerdem die Endung „.lib“.

Nun ist noch das Hauptprogramm „Proseminar.c“ aus dem obigen Beispiel zu übersetzen. Dieses enthält folgenden Quellcode:

```
#include <libProseminar.h>

int main(void)
{
	demo();
	return 0;
}
```

25 Eine ausführlichere Dokumentation und weitere Parameter sind unter http://linux.die.net/man/1/ar zu finden.

Um dieses Programm zu übersetzen muss gcc die entsprechenden Parameter für die Bibliothek „libProseminar“ übergeben werden (da hierfür keine pkg-config Informationen verfügbar sind):

```
gcc -o Proseminar Proseminar.c -lProseminar -I. -B. `pkg-config --libs --cflags glib-2.0`
```

Die Angabe für die verwendete Bibliothek im „`-l`“ Parameter ist dabei der Name der Archivdatei („libProseminar.a“) ohne die Dateiendung und ohne den Teil „lib“.

Für den Include-Pfad („`-I.`“) wird das aktuelle Verzeichnis hinzugefügt, da sich die Bibliothek im gleichen Ordner wie das Programm befindet.

Mit dem Parameter „`-B.`“ wird das aktuelle Verzeichnis auch als Standard-Such-Pfad für `gcc` hinzugefügt.

Zusätzlich werden noch die Parameter für die GLib mit `pkg-config` eingefügt.

Die Übersetzung des Programms sollte nun ohne Fehler durchgeführt werden können. Ein Aufruf des Programms in der Kommandozeile bewirkt das gewünschte Ergebnis:

```
ludwig@ludwig-pc:~/CProg/Proseminar$ ./Proseminar
Benutzen der GLib Funktion g_printf()
```

4. Quellen

Literatur:

Helmke/Isernhagen, Softwaretechnik in C/C++, 2001, Hanser Verlag

Internet:

http://code.google.com/p/waf/
http://de.wikibooks.org/wiki/C-Programmierung:_Variablen_und_Konstanten
http://en.wikipedia.org/wiki/Compilation_unit
http://de.wikipedia.org/wiki/Bibliothek_%28Programmierung%29
http://de.wikipedia.org/wiki/GLib
http://de.wikipedia.org/wiki/Linker_%28Computerprogramm%29
http://de.wikipedia.org/wiki/Modul_%28Software%29
http://en.wikipedia.org/wiki/Library_%28computing%29#Static_libraries
http://en.wikipedia.org/wiki/Makefile
http://gtk.org/
http://linux.die.net/man/1/ar
http://linux.die.net/man/1/pkg-config
http://www.eclipse.org
http://www.freedesktop.org/wiki/Software/pkg-config
http://www.gnu.org/software/libc/
http://www.nag.co.uk/
http://www.w3.org/Library/