Inhalt

8 Das Konfigurationswerkzeug make

8.1 Modularisierung

Ein C-Programm muss nicht unbedingt auf einer einzigen Quelldatei vorliegen (s. Kapitel 1):

Erste Quelldatei (Modul **main.c**):

Direktiven für den Präprozessor
globale Deklarationen

int main(. . .)
{
lokale Deklarationen
Anweisungsfolge
}

Zweite Quelldatei (Modul **modul1.c**):

Direktiven für den Präprozessor
globale Deklarationen

Typ ***funktion_1*****(. . .)**
{
lokale Deklarationen
Anweisungsfolge
}

Weitere Quelldatei (Modul **modul2.c**):

Direktiven für den Präprozessor
globale Deklarationen

Typ ***funktion_2*****(. . .)**
{
lokale Deklarationen
Anweisungsfolge
}

. . .

Typ ***funktion_n*****(. . .)**
{
lokale Deklarationen
Anweisungsfolge
}

So lässt sich die Berechnung der *Exponentialfunktion* (s. Kapitel 6), wie auch die Berechnung der *Potenz*, von ihren Testprogrammen trennen. Es ergeben sich folgende **Module**:

funktionen.h *Deklarationen eigener Funktionen*

```
/*
 * Header zu funktionen.c
 */

/*
 * Exponentialfunktion (Reihenentwicklung)
 * e hoch x = 1 + x + x hoch 2 / 2! + x hoch 3 / 3! ...
 * Eingabe: x
 */
double myexp( double);

/*
 * Potenzfunktion
 * Eingabe: Basis, Exponent
 */
double mypow( float, unsigned int);
...
```

funktionen.c *Definitionen der in funktioenen.h deklarierten Funktionen*

```
/*
 * Header: funktionen.h
 */

# include "funktionen.h"  /* Funktionesdeklarationen */

double myexp( double xx)
{
  double summeAlt, summeNeu = 1, summand = 1;
  int i = 1;

  do                              /* Reihenentwicklung */
  {
    summand *= xx / i++;                  /* Summand */
    summeAlt = summeNeu;                    /* Summe */
    summeNeu += summand;
  } while(( summeNeu != summeAlt) /*&& (i < 100)*/);

  return summeAlt;
}

double mypow( float xx, unsigned int kk)
{
  double pp = 1;
  for(; kk > 0; kk--) pp *= xx;         /* Potenz x^k */
  return pp;
}
...
```

exptab.c

```
# include <stdio.h>
# include <math.h>       /* mathematische Funktionen */
# include "funktionen.h"       / * eigene Funktionen */

int main()
{
  ...
}
```

potenz.c

```
# include <stdio.h>
# include "funktionen.h"       / * eigene Funktionen */

int main()
{
  ...
}
```

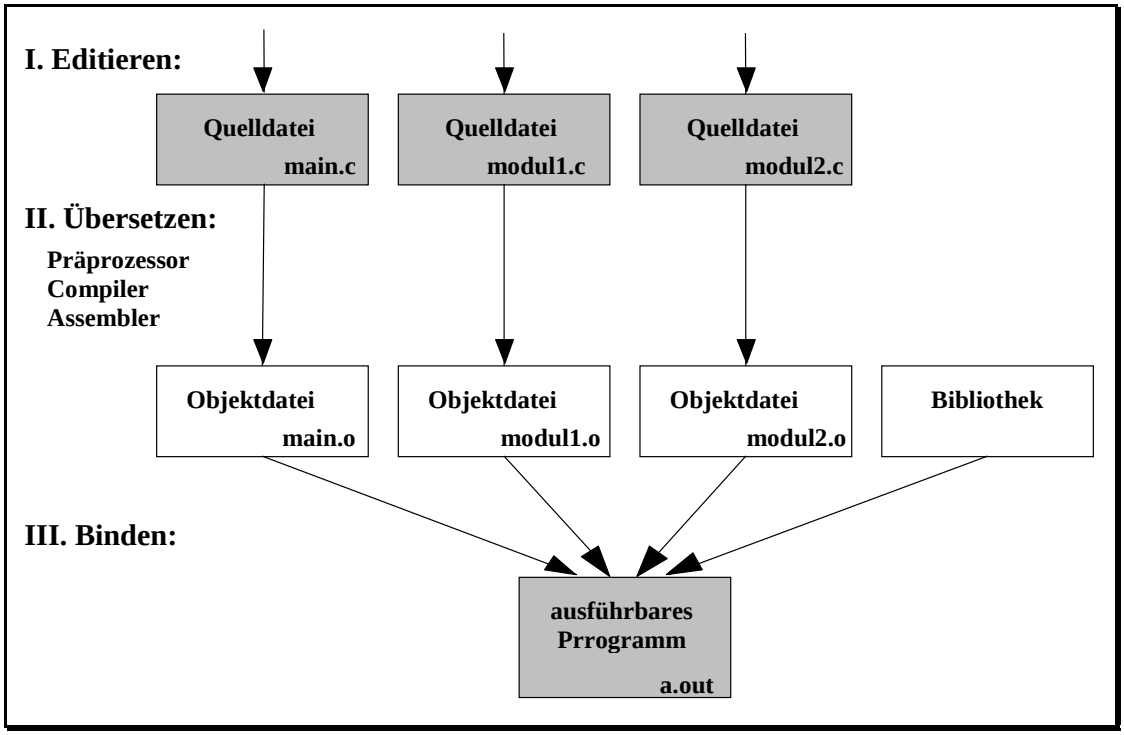
Bemerkung zu den Kommentaren

`/* Kommentar */`

In den *Deklarationen* des Headers werden Kommentare für **Benutzer**, in den *Definitionen* für **Programmierer** untergebracht.

8.2 Modulübersetzung

Bei der Erstellung eines ausführbaren Programms sind **drei Schritte** notwendig (s. Kapitel 1). Dabei wird zunächst jede Quelldatei getrennt in eine Objektdatei übersetzt. Der Binder fügt Objektdateien und Bibliotheksfunktionen zu einem ausführbaren Programm zusammen.



I. Editoren

vi	(UNIX-Standardeditor)
nedit	(UNIX-Editor mit grafischer Oberfläche)
emacs	(GNU-Editor)
xwpe	(Programmieroberfläche)

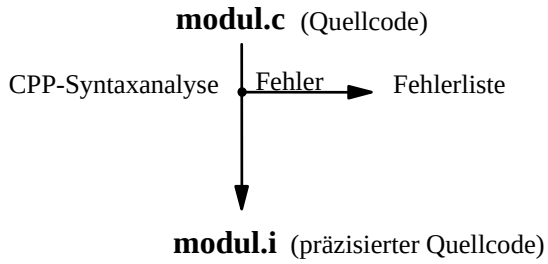
$ nedit main.c &
$ nedit modul1.c &
$ nedit modul2.c &

II. Übersetzen

Die **Übersetzung** erfolgt in 3 Phasen. Dabei entstehen Zwischencodes, welche i.R. anschließend gelöscht werden. Es besteht aber auch die Möglichkeit, und oft ist es wünschenswert, diese Zwischencodes zu behalten. Diese unterscheiden sich in ihren Endungen.

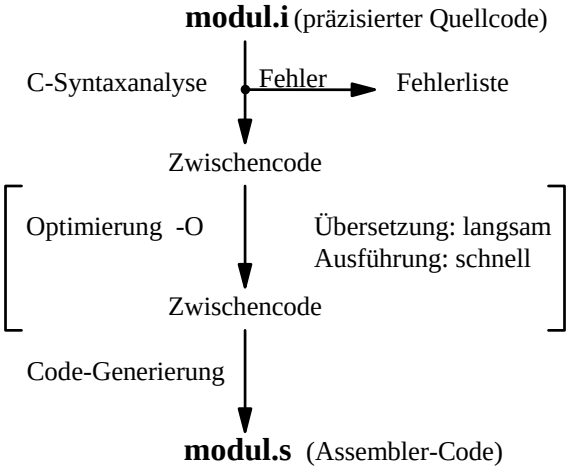
Präprozessor (1.Phase)

Präprozessordirektiven werden ausgeführt, z.B. Einfügen von ***„include“*** - Dateien und Ersetzen von ***„define“*** - Konstanten.

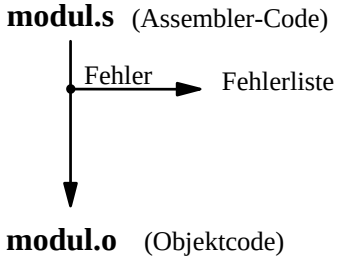


Compiler (2. Phase)

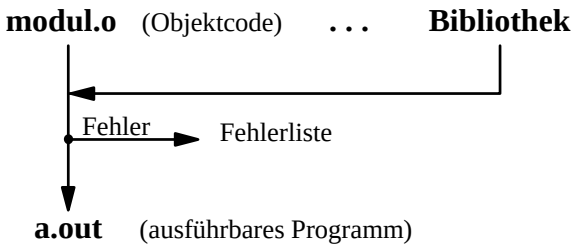
Übersetzt präzisierten Quellcode in **Assemblercode** mit Syntaxanalyse, Optimierung (wenn erwünscht) und Code-Generierung.



Assembler (3. Phase)
Übersetzt Assembler-Code in **Objektcode** (Maschinencode). Globale Größen, die in anderen Dateien deklariert sind, erhalten eine Scheinadresse (Dummy-Adresse).



III. Binden
Die Objektdateien eines Programms und benötigte Bibliotheksdateien werden zu einem **ausführbaren Programm** gebunden. Dabei werden die Scheinadressen aufgelöst.



II./III. Übersetzen und Binden
$ cc [*optionen* ***] main.c modul11.c modul22.c . . . [*** *optionen* ***]*** (UNIX-Standard)
$ gcc [*optionen* ***] main.c modul11.c modul22.c . . . [*** *optionen* ***]*** (GNU – C)

optionen ... Sie dienen der Steuerung beim Übersetzen und Binden.

Compiler:	**-P (**gcc:-**E)**	Nur Präprozessorlauf: **prog.i**.
	-S	Präprozessor- und Compilerlauf: **prog.s**.
	-c	Übersetzen ohne Binden: **prog.o**.
	-O	Übersetzen mit Optimierung.
	-ansi	Ansi-Standard
	-Wall	Alle Warnungen werden angezeigt.
	-g	Debuggerinformationen (Generierung der Symboltabellen)
	-I*include.dir*	Suchpfad für Include-Dateien (Standard: **/usr/include**).
Binder:	**-l***libery*	Bibliothek *libery* ist aus dem Standard-Bibliotheks-Verzeichnis dazuzubinden. In den Standardsuchpfaden wird die Bibliothek **lib***library***.a** gesucht. Standard-C-Funktionen **libc.a** werden stets geladen.
	-L*library.dir*	Suchpfad für Bibliotheken (Standard: **/lib** und **/usr/lib**).
	-o *name*	Das ausführbare Programm heißt *name*, sonst **a.out**.

Bei der Übersetzung muss der Quelltext aller beteiligten Dateien angegeben werden:

```
$ gcc -Wall -O -ansi exptab.c funktionen.c -lm -o exptab    =>  exptab
$ gcc -Wall -O -ansi potenz.c funktionen.c -o potenz        =>  potenz
```

Besser wäre eine getrenntes Übersetzen der einzelnen Dateien und zwar nur, falls sich der Quelltext geändert hat. Die übersetzten Dateien werden abgespeichert.

```
$ gcc -Wall -O -ansi -c funktionen.c    =>  funktionen.o
$ gcc -Wall -O -ansi -c exptab.c        =>  exptab.o
$ gcc -Wall -O -ansi -c potenz.c        =>  potenz.o
```

Anschließend können die erzeugten Objektcode der eigenen Funktionen und die verwendete mathematische Funktion aus der Bibliothek **/usr/lib/libm.a** gebunden werden.

```
$ gcc funktionen.o exptab.o -lm -o exptab    =>  exptab
$ gcc funktionen.o potenz.o  -o potenz       =>  potenz
```

Der Aufruf des **ausführbaren Programms** erfolgt dann wie üblich:

```
$ ./exptab
$ ./potenz
```

Suffix von Dateinamen in Zusammenhang mit C

.c	C source file	C-Quellcode
.h	C header (preprocessor) file	C-Header für Präprozessor
.i	preprocessed C source file	C-Quellcode nach Präprozessorlauf
.s	assembly languarge file	Assemblercode
.o	object file	Objektcode
.a	archive file	Bibliothek

8.3 Konfigurationswerkzeug make und Aufbau eines Makefiles

Die Übersetzung in Abhängigkeit von Änderungen im Quelltext übernimmt das Konfigurationswerkzeug ***make***. Um dieses Werkzeug einzusetzen, muss man den Übersetzungsvorgang verstanden haben.

make ist ein Unixwerkzeug zum Ausführen von Aktionen (wie Übersetzen, Binden, Drucken) anhand von **Zeitstempeln** und **Abhängigkeiten** zwischen verschiedenen Modulen und beteiligten Dateien. Zur Steuerung der Erzeugung eines ausführbaren Programms dient eine spezielle Datei mit dem Namen ***makefile*** und eine standardmäßig gegebene Konfigurationsdatei ***make.cfg***.

make kennt den **Zeitstempel** (Zeitpunkt der letzten Änderung) => Betriebssytem
make kennt nicht die **Abhängigkeiten** zwischen den Objektdateien => **makefile**
make kennt nicht die **Kommandos** zur Generierung eines Moduls => **makefile**

Beispiel eines ***makefile*** in einfachster Form für die Testprogramme der Exponential- und Potenzfunktion, wobei mit TAB stets die Tabulatortaste gemeint ist.

makefile

```
# makefile fuer Testprogamme exptab (e^x) und potenz (x^k)
# einfachste Form

all: exptab potenz

exptab: funktionen.c exptab.c funktionen.h
TAB gcc -Wall -O -ansi funktionen.c exptab.c -lm -o exptab

potenz: funktionen.c potenz.c funktionen.h
TAB gcc -Wall -O -ansi funktionen.c potenz.c -o potenz
```

Mögliche Aufrufe:

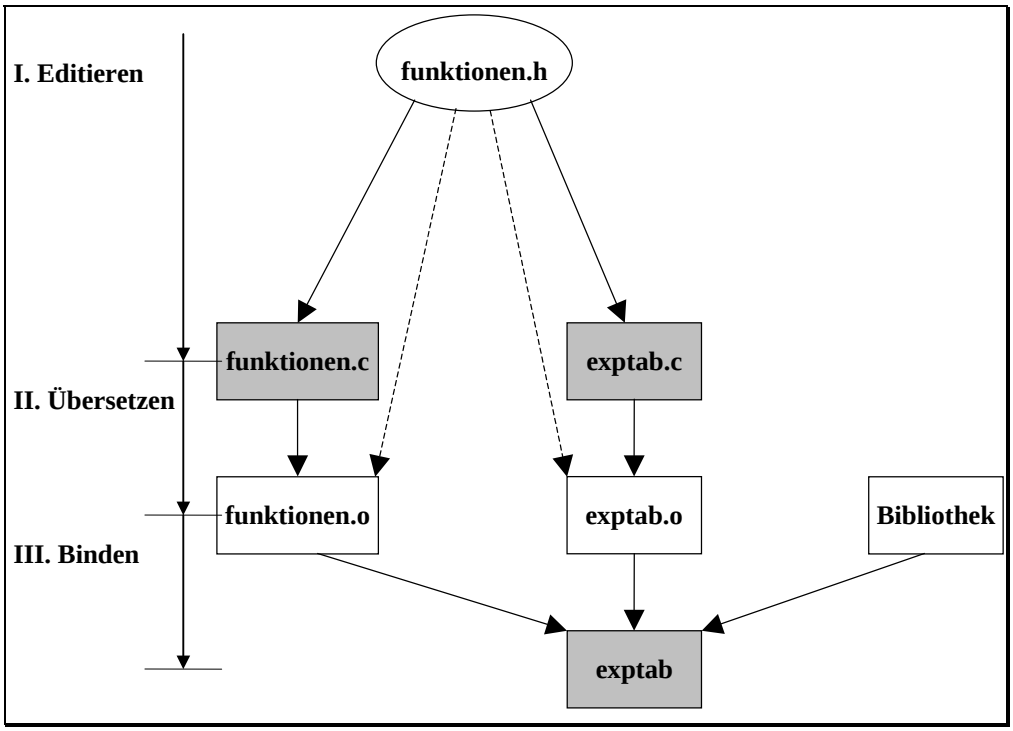
$ make	Beide Programme werden gewartet.
$ make exptab	Nur **exptab** wird gewartet.
$ make potenz	Nur **potenz** wird gewartet.

8.3.1 Abhängigkeiten und Kommandos

Ist das Programm aus mehrere Module zusammengesetzt, so ist bei der Abänderung eines Moduls nur dessen neue Übersetzung notwendig. Anschließend kann der Binder den Objektcode des geänderten Moduls mit den schon früher erzeugten Objektcodes der unveräderten Module zu einem ausführbaren Programm zusammenfügen. Dazu müssen aber die Objektdateien zur Verfügung stehen, d.h. abgespeichert werden.

Soll die Übersetzung automatisiert werden, d.h. durch ***make*** gesteuert, muss bekannt sein, welche Module von dem geänderten Modul abhängen und damit auch neu zu übersetzten sind. Die Abhängigkeiten dieser lassen sich durch einen Ableitungsbaum bestimmen.

Speziell für die Exponentialfunktion und Potenzfunktion haben wir folgendes Abhängigkeitsverhalten:



makefile für ***exptab*** und ***potenz***:

Kommentar

```
# makefile fuer Testprogamme exptab (e^x) und potenz (x^k)
# unter Beruecksichtigung von Anhängigkeit und Zeitstempel

all: exptab potenz                                  Abhängigkeiten

exptab: exptab.o funktionen.o                       Abhängigkeiten
TAB gcc exptab.o funktionen.o -lm -o exptab         Kommando

exptab.o: exptab.c funktionen.h
TAB gcc -Wall -O -ansi -c exptab.c

funktionen.o: funktionen.c funktionen.h
TAB gcc -Wall -O -ansi -c funktionen.c

potenz: potenz.o funktionen.o
TAB gcc potenz.o funktionen.o -o potenz

potenz.o: potenz.c funktionen.h
TAB gcc -Wall -O -ansi -c potenz.c

clean:
TAB rm -f funktionen.o exptab.o potenz.o
```

$ make => **gcc -Wall -O -ansi -c exptab.c** *Protokoll*
=> **gcc -Wall -O -ansi -c funktionen.c**
=> **gcc exptab.o funktionen.o -lm -o exptab**
=> **gcc -Wall -O -ansi -c potenz.c**
=> **gcc potenz.o funktionen.o -o potenz**

$ make (nur **exptab.c** wurde geändert)
=> **exptab.c** nur in Abhängigkeit von **exptab.o**
=> **gcc -Wall -O -ansi -c exptab.c**
=> **gcc exptab.o funktionen.o -lm -o exptab**

$ make potenz => **is up to date**

$ make clean => **rm -f funktionen.o exptab.o potenz.o**
(Löschen aller Objektdateien ohne Rückfrage)

8.3.2 Makrodefinitionen

Syntax: *Makroname = Zeichenfolge*
Aufruf: **$ (** *Makroname* **)**

makefile für ***exptab*** und ***potenz*** mit Makros

Kommentar

```
# makefile fuer Testprogamme exptab (e^x) und potenz (x^k)
# mit Makros

CC = gcc                                          Makrodefinitionen
CFLAGS = -Wall -O -ansi -c
LDFLAGS = -lm
SRC = exptab.c potenz.c funktionen.c
OBJ = exptab.o potenz.o funktionen.o
OUT = exptab potenz
FILES=$(SRC) funktionen.h makefile                    geschachtelt

all: $(OUT)

exptab: exptab.o funktionen.o
TAB $(CC) exptab.o funktionen.o $(LDFLAGS) -o exptab

exptab.o: exptab.c funktionen.h
TAB $(CC) $(CFLAGS) exptab.c

funktionen.o: funktionen.c funktionen.h
TAB $(CC) $(CFLAGS) funktionen.c

potenz: potenz.o funktionen.o
TAB $(CC) potenz.o funktionen.o -o potenz

potenz.o: potenz.c funktionen.h
TAB $(CC) $(CFLAGS) potenz.c
```

```
clean:
TAB rm -f $(OBJ)

# Druckkommando, $? geänderte Dateien
# -l72 Seitenlaenge, -n mit Zeilennummerierung
# touch setzt Zeitstempel der Datei print der Laenge 0

print: $(FILES)
TAB print -l72 -n $? | lpr
TAB touch print
```

$ make print druckt alle nach dem letzten Druck veränderte Dateien

8.3.3 Ein letztes Beispiel

Ein weiteres Beispiele für ein sinnvolles Einsetzen von ***make*** sind sehr lange Kommandos, wie zum Beispiel beim Einbinden der **vogle**-Bibliothek, einer sehr einfachen C-Grafik-Bibliothek[1]:

$ gcc -Wall -O -ansi -I/dargb/pub/graf/vogle/vogle-1.3.0/local/include
kurven.c k.c -L/dargb/pub/graf/vogle/vogle-1.3.0/local/lib -lvogle -lX11 -lm
-o kurven => ***kurven***

vogle-Bibliothek **/dargb/pub/graf/vogle/vogle-1.3.0/local/lib/libvogle.a**
für Grafikfunktionen aus **/dargb/pub/graf/vogle/vogle-1.3.0/local/include /vogle.h**

makefile

```
# Dieses makefile wartet das Programm kurven.

CC = gcc
CFLAGS = -Wall -O -ansi –c
LDFLAGS = -lm –lvogle –lX11
OBJ = kurven.o k.o
LIBS = -L/dargb/pub/graf/vogle/vogle-1.3.0/local/lib
INC = -I/dargb/pub/graf/vogle/vogle-1.3.0/local/include

kurven: $(OBJ)
TAB $(CC) $(OBJ) $(LIBS) $(LDFLAGS) -o kurven

kurven.o: kurven.c k.h
TAB $(CC) $(INC) $(CFLAGS) kurven.c

k.o: k.c k.h
TAB $(CC) $(INC) $(CFLAGS) k.c

clean:
TAB rm -f $(OBJ)
```

$ make => **kurven**

[1] Die Bibliothek ist frei verfügbar auf „The Eric H. Echidna Memorial Home Page“ unter **http://bund.com.au/~dgh/eric/.**