Praxisorientierte Einführung in C++
Lektion: "Die Compiler-Chain (Vom Quellcode zum ausführbaren Programm)"

Christof Elbrechter

Neuroinformatics Group, CITEC

April 24, 2014

Table of Contents

Allgemeines

- C++ erlaubt Zerlegung des Quelltext eines Programms in Einzelteile (Funktionen, Klassen, Module, ...)
- Einzelteile bestehen i.d.R. aus 2 Teilen
- Deklaration
  - Deklariert Bezeichner und Typ (einer Funktion,..)
    - I.d.R in sogenannten "Header"-Dateien (*.h, *.hh, *.H, *.hpp, *.hxx, ...)
  - Definition = Implementation
    - In Quelltext-Dateien (*.cpp, *.cc, *.CC, *.cxx, *.C, ...)

Allgemeines

- Header-Datei deklariert Funktions- und Klassen-Interfaces (nicht zu verwechseln mit Java-Interfaces)
- Bibliotheken sind (schon fertig übersetzte) Sammlungen von Funktionialitäten
- Zusammen mit den zugehörigen Header-Dateien kann Bibliothek verwendet werden

Vom Quellcode zum Programm

- Ausgangspunkt:
  - Eine Menge von Quellcode-Dateien und Header-Dateien
- Präprozessor (Später noch mehr!):
  - Präprozessor-Anweisungen (beginnend mit #) werden expandiert (Text-Ersetzung)
  - z.B. Wird `#include <abc.h>` mit dem Inhalt der Header-Datei abc.h ersetzt
  - Mit #include können auch fremde Header-Dateien eingebunden werden
  - Wichtig und definitiv merken:
    - `#include` ist wirklich nur eine reine Textersetzung. Dabei werden keine "Packages importiert" oder ähnliches.
    - Evtl. muss noch eine Bibliothek an die ausführbare Datei "gelinkt" werden.

Vom Quellcode zum Programm

- Der Präprozessor erzeugt eine neue Datei, welche keine #-Direktiven mehr enthält
  - Aber keine Panik!: Das passiert alles implizit beim Übersetzen
- Compiler: (Übersetzer)
  - Übersetzt Quellcode in Maschinensprache
  - Output sog. Objekt-Dateien (*.o)
  - Hat nichts mit Objekten aus OO-Programmierung zu tun!

Vom Quellcode zum Programm

- Maschinencode in den Objekt-Dateien ist annotiert
  - An vielen Stellen befinden sich Referenzen auf andere Funktionen, die zwar deklariert und verwendet, aber nicht direkt definiert wurden
  - Referenzen müssen durch Spung-Befehle ersetzt werden, damit ein Programm lauffähig ist
    - Linker
  - Details: Implementationsabhängig

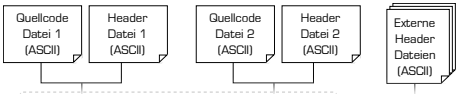
Linker

- Der Linker löst Symbol-Referenzen auf
  - Falls Symbole nicht aufgelöst werden konnten: Linker-Fehler
  - Linker erzeugt:
    - Ausführbare Programme oder
    - Statische Bibliotheken oder
    - Dynamische Bibliotheken

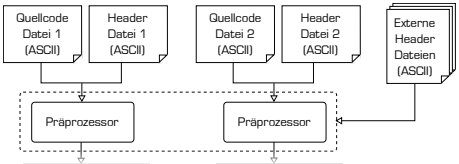
Translation Units

- I.d.R. gilt eine Quellcode-Datei als ein sog. Translation-Unit
- Diese können einzeln übersetzt werden
- Die daraus resultierenden Objekt-Dateien werden dann in einem sep. Linker-Schritt zusammen-gelinkt

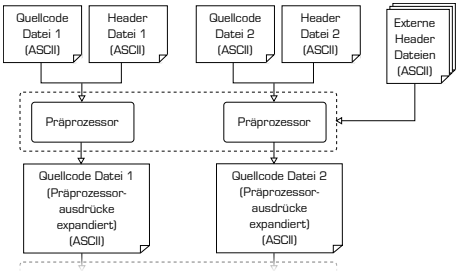
Präprozessor, Compiler und Linker



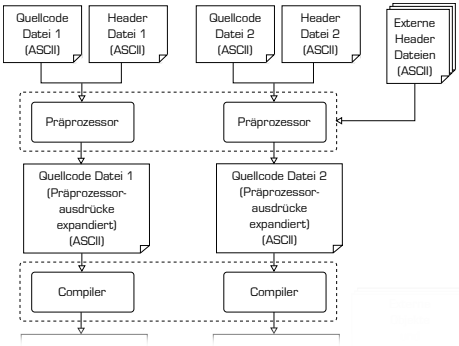
Präprozessor, Compiler und Linker



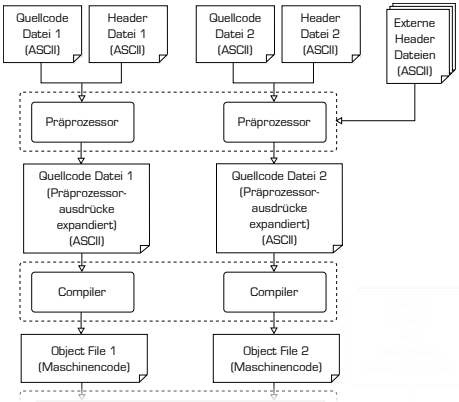
Präprozessor, Compiler und Linker



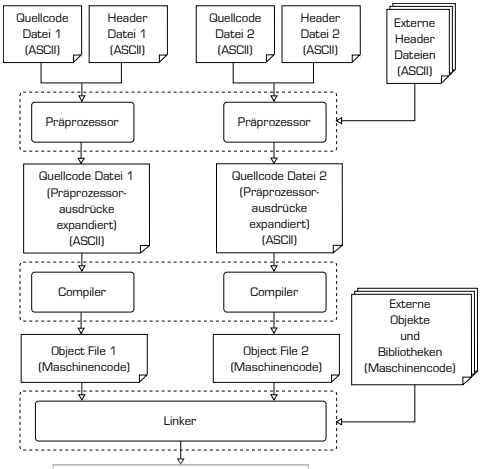
Präprozessor, Compiler und Linker



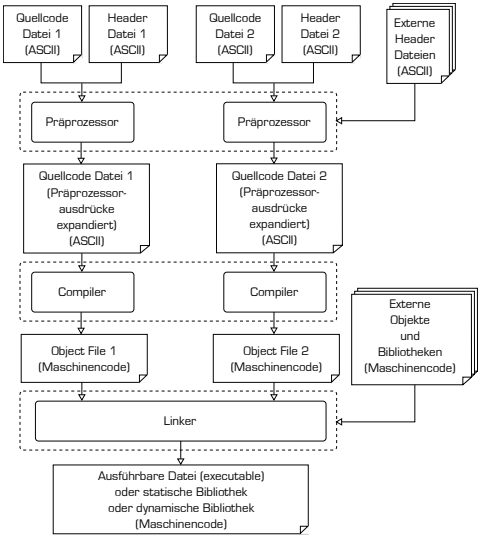
Präprozessor, Compiler und Linker



Präprozessor, Compiler und Linker



Präprozessor, Compiler und Linker



Header

- Datei "utils.h"

```
#ifndef UTILS_H
#define UTILS_H

int add(int a, int b);

void show(int i);

#endif
```

Quellcode-Datei

- Datei "utils.cpp"

```
#include <utils.h>
#include <iostream>

int add(int a, int b){
  return a+b;
}
void show(int i){
  std::cout << "this is an int:"
  << i << std::endl;
}
```

Hauptprogramm

- Datei "prog.cpp"

```
#include <utils.h>

int main(){
  int a = 5,b=6;
  int c = add(a,b);
  show(c);
  return 0;
}
```

Kompilieren mit g++

- Die "lange" Version

Shell

```
> ls
prog.cpp utils.cpp utils.h
> g++ -I. -c -o utils.o utils.cpp
> g++ -I. -c -o prog.o prog.cpp
> g++ -o prog prog.o utils.o
> ./prog
this is an int:11
```

Kompilieren mit g++

- Die "kurze" Version

Shell

```
...oder in einem Rutsch
> g++ -I. -o prog prog.cpp utils.cpp
```

Kompilieren mit g++

- Oder: Erst eine statische Bibliothek bauen und dann linken:

Shell

```
...oder statische Bibliothek 'utils.a'
> g++ -I. -c -o utils.o utils.cpp
> ar -rs utils.a utils.o
> g++ -I. -o prog prog.cpp utils.a
```

Kompilieren mit g++

Shell

```
...oder 'shared' Bibliothek 'libutils.so'
> g++ -I. -shared -fPIC -o libutils.so utils.cpp
> ls
utils.h utils.cpp prog.cpp utils.so
> g++ -I. -L. -lutils -o prog prog.cpp
> LD_LIBRARY_PATH=. ./prog
```

- Die Details sind plattformspezifisch

Mehr dazu...

- Mehr dazu in den Übungen!!