§4 Programmstruktur - Werkzeuge

Leitideen: *Getrenntes Übersetzen und nachfolgendes Linken wird meistens mit Hilfe von Werkzeugen (z.B. make) durchgeführt. Bei Änderungen werden nicht alle Dateien neu übersetzt und gelinkt, sondern nur die tatsächlich erforderlichen Aktionen vorgenommen. Die Abhängigkeiten der Programmteile und die auszuführenden Aktionen werden vom Benutzer in einer Steuerdatei (makefile) angegeben.*

Der C/C++-Präprozessor hat mehrere Arbeitsphasen, in denen unter anderem Dateien eingefügt und Textersatz vorgenommen werden. Zum Leistungsumfang gehören auch das bedingte Übersetzen und die Makroexpansion (#define mit Parametern).
In C++ beschränkt sich seine Verwendung in der Regel auf (System)headerdateien.

§4 Werkzeuge - Themenübersicht

- make - Überblick
- make - Abhängigkeiten
- C/C++-Präprozessor I - Arbeitsphasen
- C/C++-Präprozessor II - wichtigste Anweisungen
- C/C++-Präprozessor III - assert
- C/C++-Präprozessor IV - Caveat

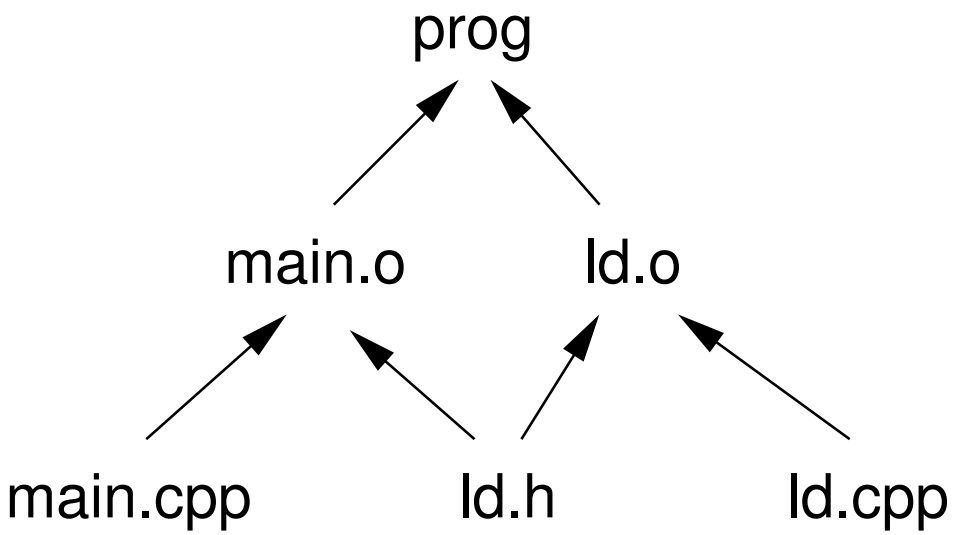
make - Überblick

Ziel: Bei Programmänderungen nicht alle Dateien neu übersetzen, sondern nur erforderliche Aktionen vornehmen

- *Z.B.:* Neu übersetzen, wenn Quellcodedatei jünger als Objektcodedatei (Vergleich der Zeitstempel)
- *Deshalb:* Kenntnis der „Abhängigkeiten“ (dependencies) notwendig
- Beschreibung der Abhängigkeiten und auszuführenden Aktionen in einer Steuerdatei („makefile“)
- `make` liest makefile und führt notwendige Aktionen aus
- makefile-Syntax verschieden von Shellscriptsyntax:
  - Tab muss Aktion einleiten (Ersatz durch Leerzeichen unzulässig!)
  - Variablenzugriff: $(name) anstelle von $name

make - Abhängigkeiten

Beispiel:



Makefile:

```
prog: main.o ld.o
Tab c++ main.o ld.o -o prog
main.o: main.cpp ld.h
Tab c++ -c main.cpp
ld.o: ld.cpp ld.h
Tab c++ -c ld.cpp
```

C/C++-Präprozessor

Arbeitsphasen

1. Ersetzen bestimmter Dreizeichenfolgen, z.B. `??)` $\rightarrow$ `]`
   Zweck: Eingabemöglichkeit für evtl. auf der Tastatur nicht vorhandene Zeichen
2. Verbinden aufeinanderfolgender Zeilen, falls vorherige mit `\` endet
   Zweck: Aufteilung zu langer Zeilen
3. Ausführen der Präprozessoranweisungen
   Zweck: Transparente Behandlung von Maschinenabhängigkeiten, Erhöhung der Portabilität, Import von Deklarationen (getrenntes Übersetzen), bedingtes Übersetzen
4. Auswerten der Ersatzdarstellungen in Zeichen(ketten)konstanten, Verbinden benachbarter Zeichenkettenkonstanten
   Zweck: Aufteilung langer Zeichenkettenkonstanten
5. Zerlegen in C++-Eingabesymbole
   Zweck: Input für den eigentlichen C++-Compiler

C/C++-Präprozessor II

Wichtigste Präprozessoranweisungen

- `#include <...>` Einfügen von Systemheaderdateien
- `#include "..."` Einfügen eigener Headerdateien
- `#define N ...` Symbolische Konstante definieren (Textersatz)
- `#ifdef ...` `#else` `#endif` Bedingtes Übersetzen (Maschinenabhäng.., Debuggen)

Allgemeine Anmerkungen

- C/C++-Präprozessor erhebl. komplexer als hier dargestellt: z.B. rekursive Makroauswertung, Verkettung von Präprozessoreingabesymbolen, Makros für Zeilennummern, Dateinamen, Fehlermeldungen
- Einsatz auch an unerwarteten Stellen: etwa zur Realisierung variabler Argumentlisten wie bei `printf` (`<cstdio>`)
- Einsatz auch außerhalb von C/C++, aber andere Präprozessoren besser geeignet (z.B. `m4`)

C/C++-Präprozessor III

Assertions *(nicht auf Infoblatt!)*

Zweck: Abschaltbare Debuganweisungen, mit denen Annahmen an bestimmten Stellen im Programm überprüft werden können

```
#include <cassert>     // erforderlich!

int main()
{
   // Rechnung, bei der immer x>0 erwartet wird
            :
   assert(x>0); // bewirkt Abbruch mit Fehler-
            :   // meldung, falls x<=0
```

- Deaktivieren der Assertions durch `#define NDEBUG` *vor* `#include <cassert>`

C/C++-Präprozessor IV

Caveat

- Jede Präprozessoranweisung in eine eigene Zeile! (Verteilung auf mehrere Zeilen möglich, falls Zeile mit `\` endet)
- „Unerklärliche“ Compilerfehler evtl. durch fehlerhafte Präprozessoranweisungen verursacht (ggf. nur Präprozessorlauf durchführen, z.B. bei g++: `c++ -E`)
- Zwei klassische Fehler:

```
#define N=100
```
Gleichheitszeichen falsch

```
#define sqr(x) x*x
```
Klammern fehlen

- Sprachelemente von C/C++ vorrangig vor Präprozessoranweisungen verwenden:

```
#define a(i,j) a[N*i+j]
double a[N*N];
```
auch in C keine gute Idee für Matrixoperationen

```
#define sqr(x) ...
```
Inlinefunktion besser

```
#define max(a,b) ...
```
Funktionstemplate besser