Programmieren in C++

SS 2011

Vorlesung 2, Mittwoch 11. Mai 2011

(Compiler und Linker, .h und .cpp Dateien)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Feedback zu Ihren Erfahrungen
- Compiler und Linker
  - Was macht der Compiler?
  - Was macht der Linker?
  - Warum die Trennung in .h und .cpp Dateien?
  - Konsequenzen für das Makefile
- Continuous Build System
  - Was ist das? Wie hilft uns das?
- Vim
  - Kurze Einführung

Ihre Erfahrungen bisher

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Das Drumherum hat am meisten Zeit gekostet
    - Das Einrichten (von Linux) überhaupt
    - Installation von Google Test, -lpthread Problem
    - Verständnisprobleme mit SVN
    - Probleme bei der Benutzung von vim
  - Probleme bei ASSERT_EQ mit Fließkommazahlen
  - cpplint.py ist pingelig, nervig, lästig, etc.
  - Letzte Vorlesung gegen Ende etwas gehetzt / chaotisch
  - Zeitaufwand 20 Minuten bis 10 Stunden
  - Forum wurde intensiv genutzt, sehr schön!
  - Herzlichen Dank für ihr ausführliches Feedback!

Compiler und Linker

UNI FREIBURG

- Warum die Unterscheidung
  - Eigentlich will man ja nur ein lauffähiges Programm, und dafür ist der Compiler da, warum also so kompliziert?
  - Grund: Code ist oft sehr umfangreich und man ändert ihn inkrementell
    - dann möchte man nur die Teile neu kompilieren müssen, die sich geändert haben!
    - insbesondere will man ja nicht jedesmal die ganzen Standardfunktionen (wie z.B. printf) neu kompilieren

Am Beispiel vom BirthdayParadox Code

UNI FREIBURG

- 1. Schritt (Compiler)
  - Wir übersetzen die Funktion, das Main und das Test separat, aber machen noch kein ganzes Programm daraus
    - das geht mit g++ -c <Name der .cpp Datei>
    - damit bekomme wir zu jeder .cpp Datei eine .o Datei
    - die enthält den Maschinencode für die jeweiligen Funktionen
    - mit nm –C <Name der .o Datei> sieht man welche Funktionen eine .o Datei bereitstellt (T) und welche sie von woanders benötigt (U)

Am Beispiel vom BirthdayParadox Code

UNI FREIBURG

- 2. Schritt (Linker)
  - Wir fügen die .o Dateien zu einem einzelnen ausführbaren Programm zusammen, das nennt man linken
    - BirthdayParadox.o und BirthdayParadoxMain.o geben das ausführbare Main Programm
    - BirthdayParadox.o und BirthdayParadoxTest.o geben das ausführbare Test Programm
  - Beim Zusammenfügen muss gewährleistet sein
    - dass jede Funktion, die in einer der gelinkten .o Dateien benötigt wird von genau einer anderen bereitgestellt wird
      - sonst "undefined reference" bzw. "multiple definition of"
    - dass genau eine main Funktion bereitgestellt wird
      - sonst "multiple definition of main"

Am Beispiel vom BirthdayParadox Code

UNI FREIBURG

- 2. Schritt (Linker) ... Fortsetzung
  - Eine Funktion bereitstellen, die nirgendwo benötigt wird ist kein Problem und gibt auch keine Fehlermeldung
    - insbesondere machen das die Standardbibliotheken in hohem Maße
    - eine Bibliothek ist nichts anderes eine .o Datei
      - mit einer typischerweise großem Menge an bereitgestellten Funktionen
      - und eine speziellen Index, so dass der Linker die gewünschte Funktion schnell findet
      - mehr zu Bibliotheken auf den nächsten Folien ...

Bibliotheken 1/3

UNI FREIBURG

- Wissenswertes
  - Standardbibliotheken oder solche von anderen Programmen stehen typischerweise im Verzeichnis /usr/lib oder /usr/local/lib
  - es gibt statische und dynamische Bibliotheken → nächste Folie
  - sie heißen lib<name>.a (stat.) bzw. lib<name>.so.<x> (dyn.)
  - Man kann Sie einfach über ihren Dateinamen dazulinken, z.B.

    g++ BirthParadoxTest.o BirthdayParadox.o /usr/lib/libgest.a
  - Besser ist aber man schreibt

    g++ BirthdayParadoxTest.o BirthdayParadox.o –lgtest
    - der Compiler sucht dann in seinen Standardverzeichnissen (siehe oben) nach libgtest.a bzw. libgtest.so.1 etc.
    - mit –L <Ordnername> kann man angeben, wo noch gesucht werden soll

Bibliotheken 2/3

UNI FREIBURG

- Statisch vs. Dynamisch
  - Code aus einer statischen Bibliothek wird Teil des ausführbaren Programms
    - Vorteil: man braucht die Bibliothek nur beim Linken aber nicht zum Ausführen des Programmes
    - Nachteil: das ausführbare Programm kann sehr groß werden
  - Bei einer dynamischen Bibliothek steht im ausführbaren Code nur eine Referenz auf die Stelle in der Bibliothek
    - Vorteil: das ausführbare Programm wird kleiner
    - Nachteil: man braucht die Bibliothek zur Laufzeit
    - Mit ldd bekommt man die von einem ausführbaren Programm benötigten dynamischen Bibliotheken

Bibliotheken 3/3

UNI FREIBURG

- Man kann sich Bibliotheken auch leicht selber bauen
  - Eine statische Bibliothek baut man einfach (wie eine .o Datei auch) mit g++ -c –o lib<name>.a ...
  - Mit ar c lib<name>.a <.o file 1> <.o file 2> ... kann man sich aus einer beliebigen Menge von .o Dateien eine statische Bibliothek bauen
  - Eine dynamische Bibliothek baut man mit g++ -fpic –shared –o lib<name>.so ...
    - Das ausführbare Programm sucht dann **zur Laufzeit** in den Standardverzeichnissen nach dieser Bibliothek
    - Steht die Bibliothek woanders muss man setzen

      export LD_LIBRARY_PATH=<folder name>

Header bzw. .h Dateien

UNI FREIBURG

- Wofür braucht man die?
  - Bevor man eine Funktion benutzt, muss man sie deklarieren (so wie eine Variable)
    - auch wenn die Implementierung in einer anderen Datei steht (und dann am Ende dazugelinkt wird)
  - Eine .cpp Datei (oder eine ganze Bibliothek) stellen in der Regel viele Funktionen zur Verfügung
    - jeder, der eine oder mehrere von diesen Funktionen benutzen will muss sie dann erst deklarieren
    - deswegen sammelt man die ganzen Deklarationen in einer sogenannten header Datei, die enden auf .h
    - braucht man eine oder mehrere davon macht man

      #include <xyz.h> sucht in /usr/include etc. oder

      #include "./xyz.h" Pfad explizit angeben

Header guards

UNI FREIBURG

- Eine .h Datei kann andere .h Dateien includen
  - Bei komplexerem Code ist das sogar die Regel
  - Dabei muss man einen "include cycle" verhindern
    - Datei xxx.h included (unter anderem) Datei yyy.h
    - Datei yyy.h included (unter anderem) Datei zzz.h
    - Datei zzz.h included (unter anderem) Datei xxx.h
      - an dieser Stelle muss man verhindern, dass man xxx.h nochmal liest, sonst geht es immer so weiter
  - Dazu gibt es die sogenannten header guards am Anfang und Ende jeder .h Datei

```
#ifndef VORLESUNGEN_VORLESUNG_1_XXX_H_
#define VORLESUNGEN_VORLESUNG_1_XXX_H_
...
#endif  // VORLESUNGEN_VORLESUNG_1_XXX_H_
```

Make

UNI FREIBURG

- Abhängigkeiten
  - Man kann make sagen, dass ein bestimmtes target von anderen targets abhängt, letztere heißen dann dependencies

    ```
    <target>: <dependency 1> <dependency 2> ...
        <command 1>
        <command 2>
        ...
    ```

  - Jetzt wird bei make <target>, erstmal make <dependency 1>, make <dependency 2> usw. ausgeführt und dann erst die Kommandos <command 1>, <command 2> usw.
  - Wenn <target> ein Dateiname ist, werden die Kommandos nur ausgeführt wenn
    - es eine Datei mit dem Namen noch nicht gibt
    - oder es eine Datei mit dem Namen gibt, die aber älter ist als eine existierende Datei mit Namen <dependency i>

Continuous Build System (Jenkins)

UNI FREIBURG

- Features
  - Läuft auf einem separaten Rechner
  - "Baut" Ihr Programm periodisch oder wann immer Sie etwas im SVN ändern
    - make build
    - make test
    - make lint
    - make clean (ab dieser Vorlesung)
  - Über ein Web Interface können Sie sich die einzelnen "Builds" bequem anschauen
  - Bei Problemen bitte Mail an Jens und Axel

Vim

UNI FREIBURG

- Das ist der Editor den ich benutze
  - Sie können aber irgendeinen Editor benutzen
  - Es sollte aber einer sein, der viel kann, insbesondere
    - Syntax Highlighting
    - Autovervollständigung
    - Blick auf mehrere Dateien gleichzeitig
    - Komfortables Wechseln zwischen Dateien
  - Wenn Sie noch keine Präferenz haben und / oder aus anderen Gründen gerne Vim benutzen wollen, gebe ich Ihnen gerade eine kleine Einführung

Literatur / Links

- Compiler und Linker
  - Online Manual zum g++ Version 4.5

    http://gcc.gnu.org/onlinedocs/gcc-4.5.0/gcc/
  - Linker Optionen von eben diesem

    http://gcc.gnu.org/onlinedocs/gcc-4.5.0/gcc/Link-Options.html#Link-Options
  - Wikipedias Erklärung zu Compiler und Linker

    http://en.wikipedia.org/wiki/Compiler

    http://en.wikipedia.org/wiki/Linker_(computing)
  - Statische und dynamische Bibliotheken

    http://en.wikipedia.org/wiki/Library_(computing)