IFMP Bibliotheken

- Bibliotheken für Sachen wie ifm::integer
- **Datei “Makefile” muss (!) im gleichen Ordner sein, wie source code, den ihr kompilieren wollt**
  - sorgt dafür, dass Compiler weiss, wo er Bibliotheken findet
  - Einfach aus “unterlagen” oder “progs/lecture” heraus kopieren
- Emacs so eingerichtet, dass “Makefile” benutzt wird
  - Datei XY.cpp wird mit Befehl “make XY” kompiliert
  - “make XY” kann man auch direkt im Terminal eingeben
- Tests: “power8_exact.cpp”, “snowflake.cpp”

VirtualBox (openSUSE)

- 4 Programme, die ihr verwendet werdet
  - Emacs: Text-Editor
  - Terminal: Um Programme auszuführen
  - Dolphin: Dateien/Ordner verwalten
  - Firefox: Web browser
- Ordner:
  - progs: Schreibt eure Programme dort
  - shared: Datenaustausch VirtualBox <-> Host

Emacs

- Syntaxhervorhebung
- Code (meistens) automatisch eingerückt
- Nützliches mit F-Tasten:
  - F6: Rückt ganzen code ein (wird dadurch lesbarer)
  - F7/F8: Springt zu vorherigem/nächstem Fehler
  - F9: Übersetzt code in ausführbare Datei
  - F10: Springt zu gegebener Zeile

Terminal

- Die wichtigsten Befehle:
  - ls: Inhalt vom aktuellen Ordner
  - pwd: Pfad vom aktuellen Ordner
  - cd <Ordner>: setze aktuellen Ordner auf <Ordner>
    - kann auch relativ zum aktuellen Ordner sein (z.B. wir sind in “/X” und mit “cd Y” landen wir in “/X/Y”)
  - ./<Datei>: Führt <Datei> im aktuellen Ordner aus

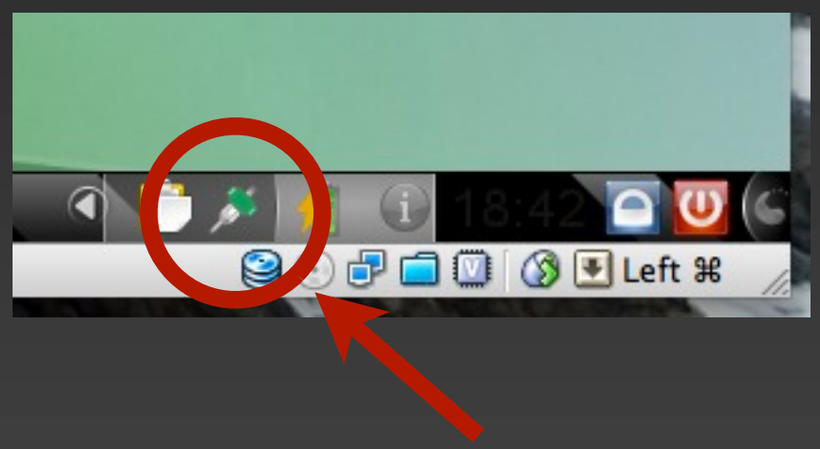
- Im Allgemeinen kann man Dateien und Ordner mittels tab-Taste automatisch vervollständigen

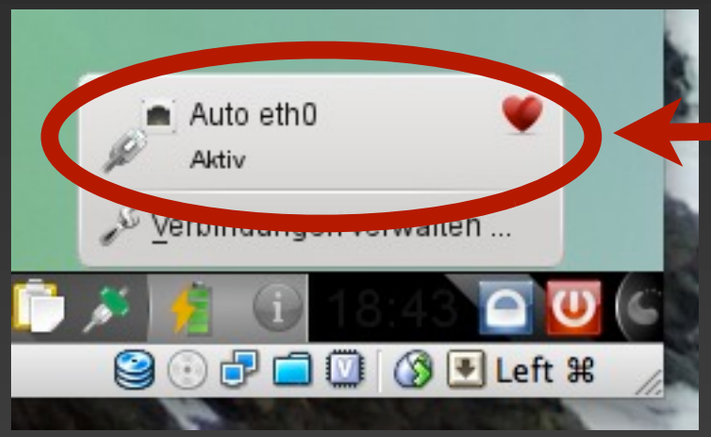
Eigenes Programm ausführen

- Ziel: XY.cpp (in progs) übersetzen und ausführen
- Emacs: Code mit F9 übersetzen (ergibt Datei XY)
- Terminal öffnen: (aktueller Ordner ist /home/ifmp10)
  - Verzeichnis wechseln: “cd unterlagen/progs” (oder “cd /home/ifmp10/unterlagen/progs”)
    - shortcut: Terminal von Delphin aus öffnen mit “Extras/Terminal öffnen” (oder Shift-F4)
  - Datei ausführen: “./XY”

Netzwerk Probleme

- Normalerweise sollte man in der VirtualBox ins Internet kommen, wenn das im Host geht
- Host sollte mit Netzwerk verbunden sein, wenn VirtualBox gestartet wird (nicht erst nachher)
- Sonst kein Internet in der Box
- Symbol unten rechts zeigt Verbindungsstatus





Klicken und warten...

Compiler Details

- Source code (e.g. “XY.cpp”) ist C++ code
  - Header-Dateien (e.g. “XY.h”) ist auch C++ code, aber üblicherweise nur Deklarationen dort

- Objekt-Datei (e.g. “XY.o”) ist kompilierter source code (inklusive eingefügten (#include) Headers)

- Bibliotheken (e.g. “libXY.a”) sind Sammlungen von vor-kompilierten codes (also viele Objekt-Dateien)
  - dazu da, dass man nicht immer alles neu kompilieren muss

Compiler Details

- Arbeitsweise eines Compilers:
  - Input: Source code “XY.cpp”
  - Übersetzung (compilation) in Objektdatei “XY.o” (Maschinensprache)
    - dies enthält auch alle eingefügten (#include) Headers
      - Compiler muss wissen, wo es die Headers findet
  - Verknüpfung (linking) von Objektdateien mit externen Bibliotheken
    - Standardbibliotheken werden automatisch verknüpft
    - alles andere muss man dem Compiler angeben (oft mühsam...)
  - Ergebnis nach dem linking: ausführbare Datei “XY”

Compiler Details (g++)

- g++ (GNU C++ Compiler)
  - Standard-Compiler, auf jedem Linux/Mac installierbar
  - Unter Windows mit cygwin installierbar
  - "`g++ XY.cpp -o XY`" erstellt Programm XY bei source code XY.cpp
  - Braucht weitere Optionen, um externe libaries zu benutzen

- `-I<PFAD>`: #include sucht Datei auch in diesem Pfad
  (Fehler bei "#include <bla>": "bla no such file or directory")
- `-l<BIB>`: benutzt Bibliothek "lib<BIB>.a"
  (Fehlermeldung: "undefined reference to...")
- `-L<PFAD>`: linker sucht Bibliothek auch in diesem Pfad
  (Fehlermeldung: "directory ... following -L not found" (falscher Pfad) oder "ld: library not found..." (Pfad fehlt))

Compiler Details (make)

- make (GNU make)
  - g++ Anweisungen können schnell kompliziert werden und oft will man nur Code neu kompilieren, der sich verändert hat
- Anweisung “make <TARGET>” (e.g. “make XY”)
  - Macht alles nötige, um <TARGET> zu erstellen
  - Beispiel “make XY” mit IFMP-Makefile:
    - Kompiliert “XY.cpp” (mit g++) und erstellt Programm “XY”
    - Sorgt dafür, dass alle nötigen Bibliotheken dabei sind
  - Voraussetzung: Datei “Makefile” im selben Verzeichnis, wo man “make <TARGET>” aufruft

Eigene Umgebung einrichten

- Anweisungen auf Homepage befolgen:
  http://www.ti.inf.ethz.ch/ew/courses/Info1_10/VB_linux_direct.html
  (Voraussetzungen (my guess): g++, make, X11)

- Kein emacs?
  - Datei “Makefile” aus progs in Zielordner kopieren
  - “XY.cpp” im Terminal mit “make XY” kompilieren

- Tests: (in progs-Ordner)

```
make power8_exact
./power8_exact
make snowflake
./snowflake
```

Eigene Umgebung (tweaks)

- Variablen in Makefile:
  - “IFMHOME”: Pfad, wo “libinteger” etc installiert werden (Standard: “~/IFMP”, BITTE keine Pfade mit Leerzeichen benutzen)
  - “X11HOME”: Pfad für X11 (hat Unterverzeichnisse “include” und “lib”)
    - Wichtig: “Makefile” in “libraries”, “progs” und auch Kopien anpassen!!

- X11 nicht gefunden?
  - “X11HOME” in Makefile anpassen (siehe unten) (musste das auf meinem Mac “X11HOME = /usr/X11” setzen)