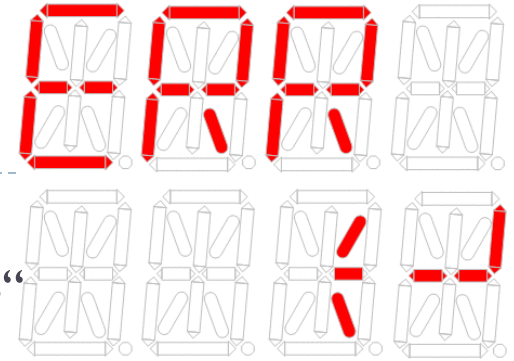
Einführung in die Programmierung
Debugging, Programmbibliotheken

Arvid Terzibaschian

Debugging

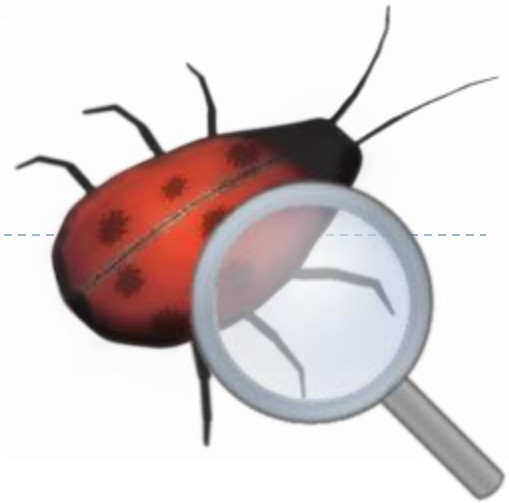
Debugging: Motivation



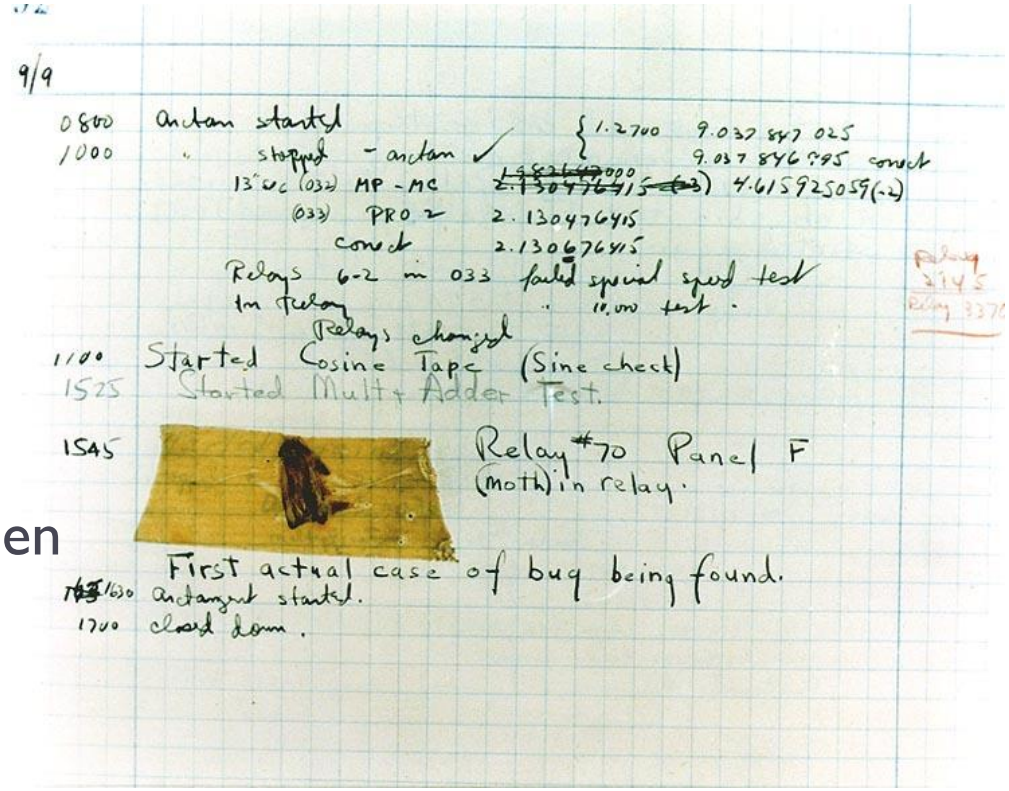
- Jedes Programm enthält Fehler
  - Anzahl steigt ziemlich konstant mit „Lines of Code“
    - im Mittel: 15-50 Fehler pro 1000 Zeilen Code*
  - nur wenige Fehler sind statisch durch Compiler feststellbar
    - meisten Fehler treten erst zur Laufzeit auf und sind abhängig vom Zustand des Programmes
- Problem: Jemand muss die Fehler finden und entfernen!

- einige Möglichkeiten:
  1. Code betrachten und Fehler „sehen“
  2. printf um Zustand von Variablen und Programm auszugeben
     - welche Werte ausgeben?
     - wann ausgeben?
  3. Debugger
     - laufendes Programm anhalten bei Fehlern
     - Zustand analysieren

Wortursprung „Debug“



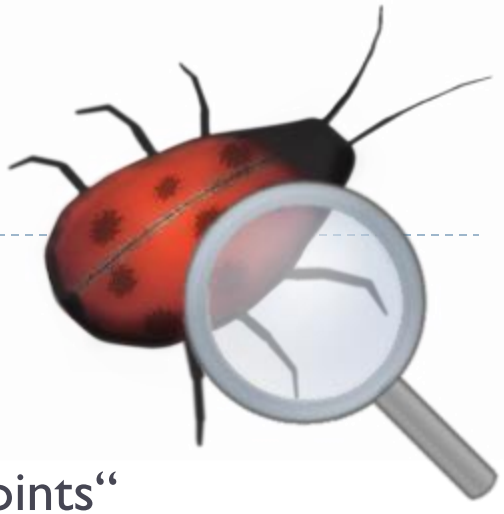
- aus dem englischen „to debug“
  - wörtlich: entkäfern
  - Bedeutung: „Fehler beseitige“
- Warum Käfer?
  - Legende besagt, das eine Motte („Bug“) im mechanischen Relais eines Computers 1947 einen Fehler auslöste. Nach dem Entfernen („Debug“) funktionierte wieder alles einwandfrei



Debugging: technischer Hintergrund

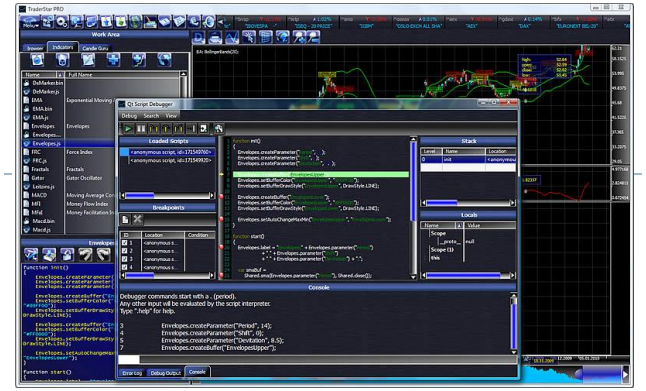
- Compiler erzeugen unlesbaren Binärcode
  - Ziel: „menschenlesbar“ den Zustand des Programms darstellen
- Lösung:
  - Compiler schreibt „Debuginformationen“ in ausführbare Datei
    - exakte Format abhängig von Compiler und Debugger
  - Debugger stellt Zustand des Programms zur Laufzeit dar
- Debuginformationen:
  - ursprüngliche Variablenbezeichnungen
  - ursprüngliche Funktionsnamen
  - Ursprungsdatei des Maschinencodes + Zeile/Spalte

Was können Debugger ?



- Hauptaufgabe:
  - Zustand von Variablen und Speicher ausgeben
  - Schrittweise Ausführung des Programms
  - automatisch Anhalten bei Haltepunkten: „Breakpoints“
    - auch mit Bedingungen
  - automatisches Anhalten bei Wertveränderung: „Watchpoints“
  - automatisches Anhalten bei „Fehlern“
    - Speicherzugriffsfehler
    - unerwartete Programmabbrüche
    - Rechenfehler z.B. Division durch 0
  - zusätzliche Features
    - Laufzeitanalyse des Programms (Runtime Profiling)
      - Z.B. mit gprof, callgrind
    - Speicherleaks entdecken (Memory Profiling)
      - Z.B. mit valgrind

Debugging in C



- Kompilieren mit „-g“
  - neben Binärcode zusätzliche Debuginformationen gespeichert
- Tool für C unter Linux„GNU Debugger“: gdb
  - arbeitet auf der Kommandozeile
  - Interaktiv, wartet auf
  - auf GDB aufbauende grafische Oberflächen
    - eclipse-cdt, MingW, Data-Display-Debugger (DDD), …
- unter Microsoft Windows: Visual C++ Debugger
  - arbeitet auch mit

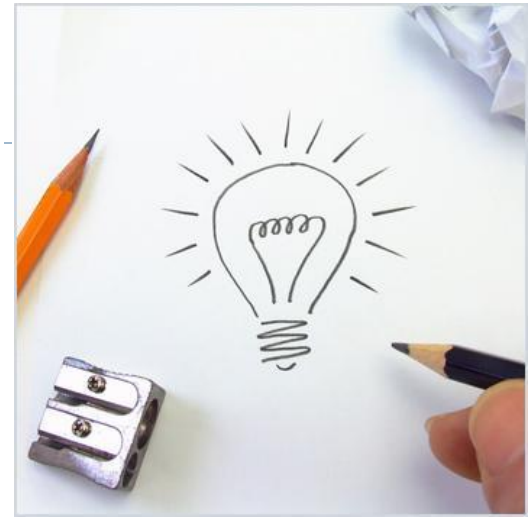
Beispiel: Debuggen mit

- DDD GUI
  für den Kommandozeilendebugger GDB
  - auch in den Pools installiert „ddd“
  - verschiedene Einführungen unter:
    - http://www.linux-magazin.de/Ausgaben/2005/04/Jagdstimmung [Deutsch]
    - http://www.linuxfocus.org/Deutsch/January1998/article20.html [Deutsch]
    - http://www.linuxfocus.org/English/January1998/article20.html [Englisch]

- Demonstration von „DDD“ jetzt in der Vorlesung
- eigenes Debugging in der Übung diese Woche
  - Gegeben: Implementierung einer Liste + Testcase
  - Ziel: Testcase zum Funktionieren bekommen!

Programmbibliotheken

Programmbibliotheken



- Hauptmotivation:
  „Das Rad nicht neu erfinden“
  - vor jeder Entwicklung prüfen ob es nicht schon eine passende Softwarebibliothek für das Problem gibt!
- Vorteile:
  - wesentlich schnellere Entwicklung
  - nur sehr wenige Bugs in vielbenutzten Bibliotheken
  - Eigene Entwicklung konzentrieren auf das Wesentliche
- Nachteile
  - Lock-in: Bibliothek später austauschen oft schwierig
  - Anpassbarkeit: Änderungen in Fremdcode selten trivial
  - u.U. Lizenzkosten

Programmbibliotheken

- mit Quellcode [Open Source]
  - Programmbibliothek in Ursprungsform in bestimmter Programmiersprache vorhanden
    - in C: .c-/.h-Dateien + Makefiles
    - kann verändert werden
    - muss selbst kompiliert werden vor Nutzung
  - u.U. schwer einzusetzen wenn eigene Entwicklungssprache oder Umgebung inkompatibel
- vorkompilierter Maschinencode [Closed Source]
  - Windows zu erkennen als *.lib/*.dll-Dateien
  - Linux *.so/*.a-Dateien
  - kein Quellcode vorhanden
  - nicht veränderbar
  - mit Interface-Dateien für unterstützte Sprachen (in C: .h-Dateien) direkt verwendbar
- Kombination von Open Source und Closed Source

Lizenzenbedingunge von Bibliotheken

- vor jeder Entwicklung prüfen, ob Lizenzbedingung von benutzten Bibliotheken passen

Lizenzname	Open-Source?	Eigenes Programm muss Bibliothekslizenz übernehmen?	Auch für kommerzielle Zwecke?	Kostenlos?
GPL	**JA**	**JA**	**JA**	**JA**
BSD, MIT, LGPL	**JA**	**NEIN**	**JA**	**JA**
Proprietär	**JA/NEIN**	**NEIN**	**JA**	**JA/NEIN**
…	**???**	**???**	**???**	**???**

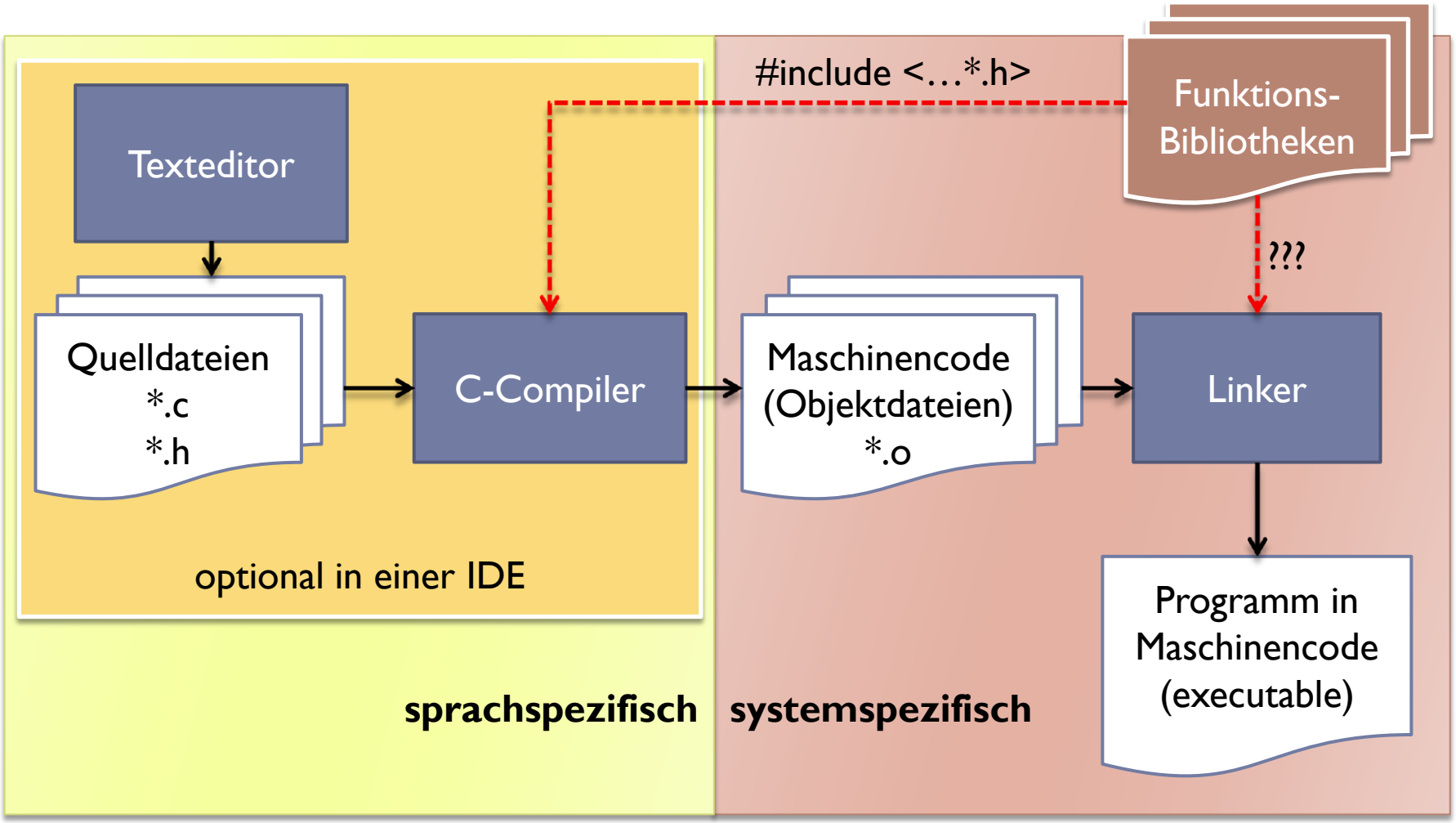
Einbinden von Bibliotheken

- Bibliothek als Sourcecode (*.c/*.h)
  - kann direkt in eigenes Programm übernommen werden
    - *.h und *.c-Dateien werden übernommen
- Vorkompilierte Programmbibliothek
  - Statische Bibliothek
    - wird über *.h-Dateien eingebunden
    - Maschinencode beim Linken mit eigenem Programmcode zu ausführbarem Programm zusammengeführt
      - Linux: *.o-Dateien/*.a-Dateien
      - Windows *.o-Dateine/*.lib-Dateien
  - Dynamische Programmbibliothek
    - wird über *.h-Dateien eingebunden
    - Maschinencode der Bibliothek bleibt eigenständige Datei
    - wird erst zur Laufzeit eingebunden
      - Linux: *.so-Dateien
      - Windows *.dll-Dateien

Einbinden von Bibliotheken

- Installation
  - oft verwendete Bibliotheken sind meist als Softwarepakete installierbar
    - z.B. unter Ubuntu z.B. mit apt-get
    - unter Windows als eigenständige Installation
  - sonst muss manuell kompiliert/installiert werden
    - Installationsprozedur unterschiedlich von Bibliothek zu Bibliothek
- Einbinden beim Erstellen des Programms
  - Erinnerung … (nächste Folie)

Erinnerung: Vom Quellcode zum Programm

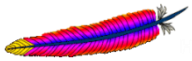


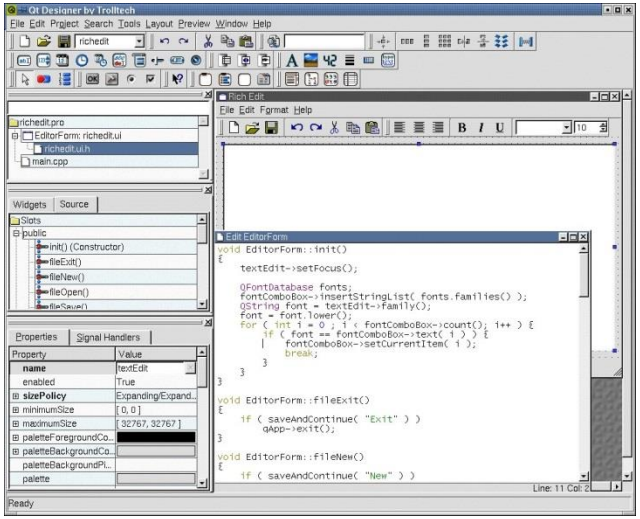
Einbinden beim Kompilieren

- Wenn Bibliothek in üblichen System-verzeichnissen installiert
  - Bibliothek heißt z.B. libopengl.so/libopengl.a
    - `gcc … -lopengl …`
  - oder: direkt Pfad zur Bibliothek angeben
    - `gcc … -l/home/arvid/opengl-2.4/libopengl.so …`

- Wenn es sich um eine dynamische Bibliothek handelt (.dll/.so) muss die Bibliothek dem Betriebssystem zur Laufzeit bekannt sein
  - üblicherweise gibt es in jedem Betriebsystem ein Verzeichnis für Bibliotheksdateien

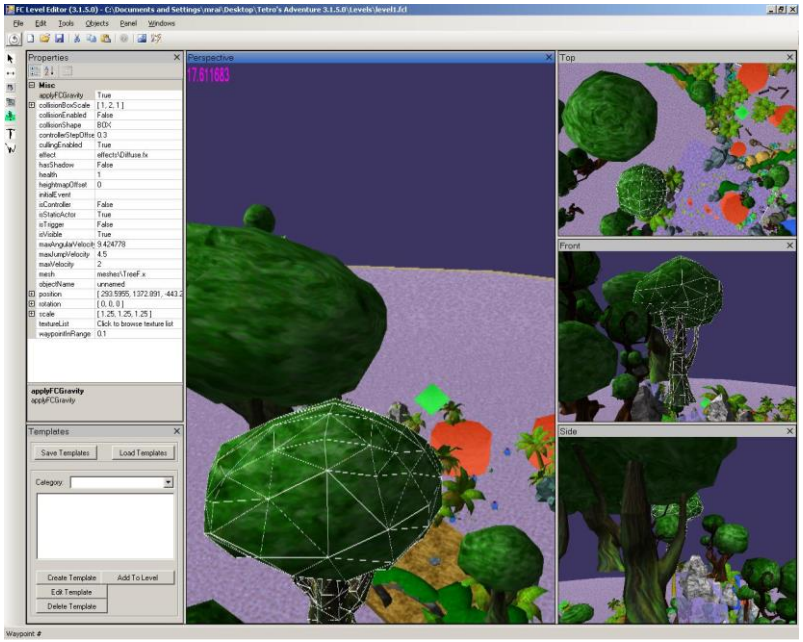
Anwendungsfelder für Bibliotheken

- **generelle Hilfe für Softwareentwickler**
  - ergänzen der Standardbibliothek
  - Apache APR (C) 
    - Container, File-IO, Threads, Memory, Hashtables, Network, User-Management
  - Boost (C++) 
    - Funktionsangebot wie Apache APR + X
- **Entwickeln von GUIs**
  - Xforms, GTK+ 
    - aufbauend auf Xwindow/Linux
  - QT (C++) 
    - Platformübergreifend
  - Win32-Library 
    - nur für Windows



Anwendungsfelder für Bibliotheken

- Multimedia: Sound und Grafik
  - OpenGL
    - Basisbausteine zur Darstellung von 3D-Szenen
  - SDL (Simple Directmedia Layer)
    - 2D/3D Darstellung, Soundausgabe Bildformate lesen, Eingabegeräte
  - FFmpeg
    - Bilder, Videos und Sound kodieren/dekodieren
  - Allegro
    - komplette Bibliothek in C zum erstellen von Spielen



Anwendungsfelder für Bibliotheken

- Mathematische/Wissenschaftliche Bibliotheken
  - BLAS/LAPACK (Fortran …)
    - lineare Algebra
    - Eigenwerte/Gleichungssysteme
    - Basis für fast alle mathematischen Programme

  - IPOpt (C++)
    - Minimieren/Maximieren von mathematischen Funktionen (Optimieren)
  - OpenCV
    - 2D- und 3D-Bildverarbeitung
    - Filtern von Bildern
    - Detektieren von Objekten in Bildern und Videos

Beispiel: Einbinden einer Bibliothek

- Wir möchten ein Face-Detection Programm schreiben
  - Anforderung:
    - Bilder sollen von Webcam kommen
    - Funktionsfähig unter Windows und Linux + Entwicklung mit C
    - in üblichen Videoaufnahmen sollte ein Gesicht erkannt werden
    - Ergebnisse sollten Live angezeigt werden
- Lösung ohne Library: ???
- Lösung mit Library:
  - kann Bilder von Webcam lesen
  - unterstützt C & Windows & Linux
  - bietet Funktion zur Gesichtserkennung
  - bietet Funktion zur Darstellung

Vielen Dank!

- Bei Fragen einfach eine Mail an:
  - arvid@cs.uni-potsdam.de