Praxisorientierte Einführung in C++
Lektion: "HelloWorld"

Christof Elbrechter

Neuroinformatics Group, CITEC

April 12, 2012

Table of Contents

Quelltext

Quelltext: "hello-world.cpp"

```
#include <iostream>
// Schreibt "Hello World!" auf die Konsole
int main(){
   std::cout << "Hello World!" << std::endl;
   return 0;
}
```

- Übersetzen mittels
  `g++ -o hello-world hello-world.cpp`
- Oder besser standard-konformen Code erzwingen mittels
  `g++ -ansi -pedantic -o hello-world hello-world.cpp`
- Ausführen mittels `./hello-world`

Separates Übersetzen und Linken

- Zunächst: Übersetzen des Programms mit C++-Compiler
  (Hier GNU gcc/g++)

```
g++ -c -o hello-world.o hello-world.cpp
```

- Daraus resultiert ein sog. Object-File (*.o)
  - Enthält Maschinen-Code (Daten u. Funktionen, allgm. Symbole)
  - **Und** Tabelle mit den Adressen hiervon
  - Sind nicht ausführbar
- Genaugenommen: `hello-world.o` enthält nur Symbole aus `hello-world.cpp`
- **Aber:** Andere verwendete Funktionalitäten fehlen noch:
  - Wie funktioniert z.B. `std::cout << ..`?
  - Wie wird der Prozess gestartet?
- $\Rightarrow$ das Programm muss **gelinkt** werden

Der Linker

- Das Object-File enthält Platzhalter für benötigte Symbole
- Linker:
  - Ersetzt(löst auf) Platzhalter durch konkrete Adressen/Sprungmarken
  - Fügt Startup-Code hinzu welcher z.B. Programm-Argumente an main übergibt
  - Erzeugt ausführbare Datei (oder Bibliothek siehe Tutorium)
- Linkeraufruf:

```
g++ -o hello-world hello-world.o
```

- In einem Schritt:

```
g++ -o hello-world hello-world.cpp
```

- Ausführbare Datei kann (unter Linux) z.B. aus der Konsole gestartet werden

```
>./hello-world
Hello World
```

Was macht man, wenn ein Fehler Auftritt?

- Programme können leicht sehr komplex werden
- Dann ist formaler Korrektheitsbeweis sehr schwierig
- Fehler schleichen sich ein
- Deswegen: Programm während der Ausführung **überwachen**
- In `hello-world` tritt kein Fehler auf, also: Bauen wir einen ein!

Quelltext: "hello-world-err.cpp"

```
#include <iostream>
int main(int n, char *args[]){
   std::cout << "program name:" << args[0] << std::endl;
   for(int i=1;i<100;++i){
      std::cout << "arg " << i << ": " << args[i] << std::endl;
   }
}
```

Verwenden eines Debuggers

- Um den Fehler zu finden, kann man natürlich *debug-Nachrichten* auf die Konsole schreiben
- Problem: Meist muss man sehr häufig (i)anpassen, (ii)übersetzen und (iii)erneut starten um den Fehler einzukreisen
- Besser: Verwendung eines Debuggers
- Kleiner Wermutstropfen: Program muss mit *Debugging Symbolen* übersetzt werden

```
g++ -g3 -O0 -o hello-world-err hello-world-err.cpp
```

- Nun kann das Program mit einem Debugger gestartet werden
- Der Debugger verwendet zusätzlich in das Program eingebaute Symbole um z.B. eine Zuordung
  Maschinencode $\Leftrightarrow$ Datei,Funktion,Quellcode-Zeile herzustellen

Debugger bieten ...

- Unterbrechen des Programmablaufs:
  - Zu beliebigem Zeitpunkt durch Userinteraktion (z.B. Tastendruck)
  - Wenn auf bestimmte Funktionen/Methoden/Variablen zugegriffen wird (Breakpoints/Watchpoints)
  - Wenn Betriebssystem Signal sendet (z.B.: SIGTERM)
- Inspektion und Manipulationvon Variablen und beliebigem Speicher im laufenden Programm
- Inspektion des Stacktrace
- Manuelles Aufrufen von Funktionen
- *Hangeln* von Programmzeile zu Programmzeile
- Evaluation von einfachen Ausdrücken
- Fortsetzen des Programmablaufs

Debuggen von "hello-world-err" mit dem GNU Debugger

```
> gdb --args hello-world-err 1 2 3 4
    [...] Reading symbols from [...]/hello-world-err...done.
(gdb) break hello-world-err.cpp:4
    Breakpoint 1 at 0x80487bd: file hello-world-err.cpp, line 4.
(gdb) run
    Starting program: [...]/hello-world-err 1 2 3 4
    program name:[...]/hello-world-err
    Breakpoint 1,main (n=5, args=0xbfffed34) at [...].cpp:4
    4 for(int i=1;i<100;++i){
(gdb) next
    5   std::cout << "arg " << i << ": " << args[i] << std::endl;
(gdb) next
    arg 1: 1
    4 for(int i=1;i<100;++i){
(gdb) ...
```

LIVE DEMO!