Einführung in C/C++

Prof. Dr. Alexander Voß
SS 2014 23.05.2014

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Ergänzungen und Erläuterungen

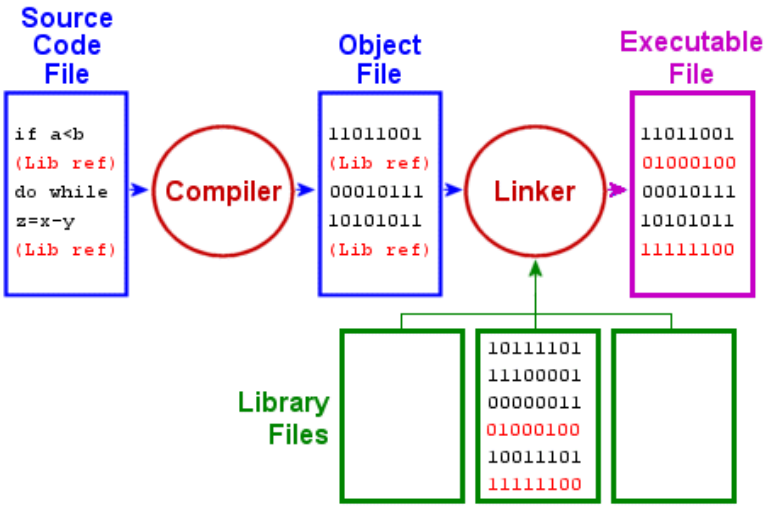
Inhalt

- Compiler und Linker
- Makefiles
- Variablen und Zeiger
- Virtuelle Tabellen

Ergänzungen und Erläuterungen

Compiler und Linker [gcc bzw. g++]

- Compiler wie gcc und g++ übersetzen C bzw. C++ Programme, d.h. Source Code-Files, in Maschinencode. Dieser findet sich nach dem Compile-Prozess normalerweise in gleichnamigen Object-Files mit Endung *.o.
- Manchmal wird der Linker (ld) implizit mit aufgerufen und es wird direkt eine ausführbare Datei generiert. Ohne weitere Angaben heisst diese hier a.out.
- Bsp. 1: Der Befehl
  ```
  gcc example.c
  ```
  generiert das Programm a.out, welches mit
  ```
  ./a.out
  ```
  ausgeführt wird.
- Bsp. 2 : Der Befehl
  ```
  gcc -c example.c
  ```
  compiliert ohne zu Linken (wg. -c) und generiert das Object-File example.o. Diese Datei kann alleine nicht ausgeführt werden. Der Aufruf von
  ```
  gcc example.o
  ```
  ruft intern den Linker auf und generiert dann wieder a.out.
- AboutDebian.Com:



- Das Object-File enthält auch schon Maschinencode, aber es sind noch nicht alle Verweise, z.B. auf Bibliotheken mit mathematischen Funktionen wie `sin`, aufgelöst. Das vollzieht der Linker und übernimmt z.B. den entsprechenden Code für `sin` aus einer Standardbibliothek mit in die ausführbare Datei.
- Es gibt 1000 Variationen vom Zusammenspiel Compiler–Linker …

Ergänzungen und Erläuterungen

Makefiles [ ]

- Das Programm `make` automatisiert den Erstellungsprozess, indem es, vereinfacht ausgedrückt, alles, was "veraltet" ist, neu generiert.
- Die Befehle und Abhängigkeiten dazu stehen in speziellen Makefiles, häufig namens `makefile`.
- Bsp. 1 für ein makefile: Die abhängigen Files sind durch ein Doppelpunkt getrennt, der Befehl steht darunter.

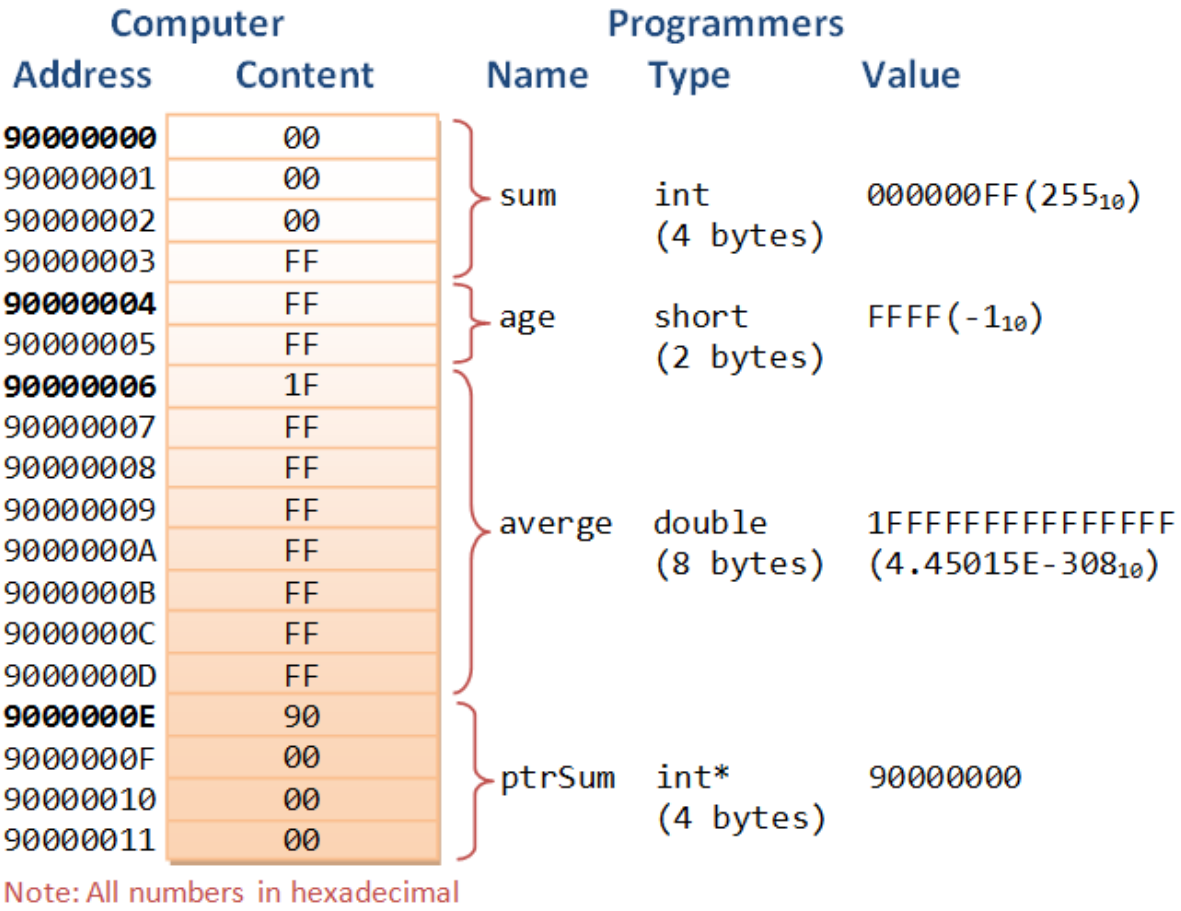
```
a.out : example.o
   gcc example.o
example.o : example.c
   gcc -c example.c
```

- Abhängigkeit bedeutet hier konkret: Wenn aus example.c das ausführbare Programm a.out erzeugt wird, so ist dieses normalerweise jünger als der Source Code, da es ja aus diesem generiert wird. Stellt man also a.out in Beziehung zu example.c (:) und ist example.c jünger als a.out, so ist an dem Source Code offenbar etwas geändert worden und a.out noch nicht erzeugt, sprich compiliert worden.
  Dazu gibt es im Makefile allgemeine Regeln bzw. Befehle, wie etwa aus *.c Files ausführbare Dateien erzeugt werden. `make` geht schliesslich alle erklärten Abhängigkeiten durch und führt diese Befehle genau dann aus, wenn abhängige Files nicht jünger sind als das, wovon sie abhängig sind.
- Es entstehen Abhängigkeitsketten, die von `make` der Reihe nach getestet und die zugehörigen Kommandos ausgeführt werden.

Ergänzungen und Erläuterungen

Variablen und Zeiger [ ]

- Lokale Variablen liegen normalerweise auf dem Stack, globale häufig in einem speziellen Datenbereich.
- Folgendes Bild gibt die Situation schön wieder: http://www3.ntu.edu.sg/home/ehchua/programming/cpp/images/Memory-AddressContent.png



Ergänzungen und Erläuterungen

Virtuelle Tabellen [ ]

- Virtuelle Tabellen werden in C++ genutzt, um dynamisches Binden zu ermöglichen. Das bedeutet, es wird auch bei einem Zeiger vom Typ Zeiger auf eine Basisklasse (Referenz entsprechend), der in Wirklichkeit allerdings ein Objekt referenziert einer abgeleiteten Klasse, die Funktion der abgeleiteten Klasse aufgerufen.
  Hier in dem Bild wäre das ein Zeiger vom Typ Hund, der auf ein Objekt vom Typ Dackel zeigt. Ein Aufruf einer virtuellen Funktion, etwa gibLaut, führt dazu, dass zur Laufzeit über die virtuelle Tabelle zunächst die richtige Funktion - die des Dackels- ermittelt und dann aufgerufen wird.
- Nicht-virtuelle Methoden weisen dieses Verhalten nicht auf. Wäre gibLaut nicht virtuell, würde in obigem Beispiel die Funktion aus Hund aufgerufen, da der Zeiger vom Typ Zeiger auf Hund ist.
- Virtuelle Methoden kosten etwas. Einerseits führen Objekte einen versteckten Zeiger mit sich, der auf die virtuelle Tabelle der Klasse zeigt. Das einzelne Objekt ist also etwas größer. Weiter wird beim Aufruf einer virtuellen Funktion immer zunächst die Funktion ermittelt. Dieser indirekte Aufruf kostet ebenfalls etwas Zeit.