Einführung in die Programmierung mit C++

Modern Family Tour: Vom Modell zum Programm

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Inhalt

Inhalt

Immer wieder freitags treffen sich Faust und Romeo mit den Jungs in einer östlich ihrer WG gelegenen Kneipe auf ein paar Bierchen.

Sie verlassen das Lokal stets um 22.30h.

Ihre (sehr verständnisvollen) Lebensabschnittsgefährtinnen Gretchen und Julia wollen sie um 23h mit dem Auto irgendwo zwischen der Kneipe und der WG abholen.

Typischerweise können Faust und Romeo dann schon nicht mehr ganz geradeaus laufen. Man beobachtet einen klaren Linksdrift, was die Vorhersage des genauen Treffpunktes für die Damen nicht einfacher macht.

Diese – clever wie sie sind – beschliessen, eine Maschine (den Computer) anzulernen, um die alkoholbelasteten Bewegungen der Herren vorherzusagen...

Sie entscheiden sich für

$$y = p \cdot x$$

als mathematisches Modell, wobei x die Ost-West-Position und y die nördliche Abweichung von der Idealstrecke bezeichnen. Julia implementiert dieses Modell als ein Programm in C++.

Der ihr leider unbekannte Parameter p quantifiziert den Einfluss der 5 Bier auf der Herren Linksdrift.

Wenn

1. das Modell der Realität entspricht und
2. die Damen den korrekten Wert für p finden können,

dann wären sie in der Lage, den genauen Treffpunkt vorherzusagen.

Problem: Wie kommt sie an p ran?

Modern Family

Suchraum

→ Tafel: (x, y)-Koordinatensystem

Im folgenden sind alle Geschlechter beliebig austauschbar, z.B. ...

©dex1.info: Dinge, die Männer an Frauen unattraktiv finden

Inhalt

Modern Family

Der Damen 1. Programm


```
1  // naumann@stce.rwth-aachen.de
2
3  #include <iostream>
4  /* standard library (stdlib) support for i/o
5     access to members of namespace via std::member */
6
7  int main() { // yields executable
8    float p=2; // memory initialized with value of real parameter
9    float x=0; // initialized memory to hold value of real state
10   std::cout << "x="; // output to screen
11   std::cin >> x; // input from keyboard
12   float y=p*x; // evaluation of real model value
13   std::cout << "y=" << y << std::endl;
14   return 0; // done (successfully)
15 }
```

→ Live: Tour_MF001.cpp

→ Tafel: Was passiert im Computer?

Beachte:

- Kommentare vorzugsweise auf Englisch
- Kontaktinformationen des Autors als Präamble jeder Quellcodedatei

Inhalt

- Installation von Virtual Box (`www.virtualbox.org/`)
- Herunterladen von VM von RWTHmoodle[1] und Entpacken
- Hinzufügen von VM zu Virtual Box und starten

Siehe auch live Demo.

Siehe auch separates Video, Globalübung und Tutorien.

[1] Alternativen: Eigene Linux Installation, Verwendung des Linux Clusters am IT Center

Inhalt

- C-Präprozessor (cpp): `g++ -E program.cpp`
- C++ Compiler (wenig Optimierung): `g++ -O0 -S program.cpp`
- C++ Compiler (mehr Optimierung): `g++ -Ofast -S program.cpp`
- Assembler: `g++ -c program.cpp`
- Binder: `g++ program.cpp` (bzw. `g++ program.cpp -o program.exe`)

Compileroptionen kennenlernen! → man g++ bzw. `gcc.gnu.org`

→ Live: Erstellung und Inspektion der Versionen

Übersetzungsprozess

Der Compiler ist dein Freund!

- Compilerfehler, z.B. Klammer vergessen ...

```
1 | int main( { ... }
```

bzw.

```
1 | int main() float ... }
```

- Binderfehler, z.B.

```
1 | int mein() { ... }
```

- Warnungen immer aktivieren (`g++ -Wall -Wextra -pedantic ...`), z.B. fehlende Initialisierung von `p` in Tour_MF001.cpp; zusätzliches Semikolon

→ Live: Fehler und Effekte

Übersetzungsprozess

Automatisierung mittels `make`

```
1  EXE=$(addsuffix .exe, $(basename $(wildcard *.cpp)))
2
3  CPPC=g++ -std=c++17
4  CPPC_FLAGS=-Wall -Wextra -pedantic -g
5
6  all : $(EXE)
7
8  %.exe : %.cpp
9          $(CPPC) $(CPPC_FLAGS) $< -o$@
10
11 clean:
12         rm -f $(EXE)
13
14 .PHONY: all clean
```

→ Live: Verwendung von `make`

- Makefile
- Verwendung des C++17 Standards
- *debug* Symbole
- Übersetzung aller Programme durch Eingabe von `make`
- Löschen aller generierter Dateien durch Eingabe von `make clean`
- Abhängigkeiten
- automatische *Ziele*
- unechte *Ziele*

Inhalt

Weitesgehende Überprüfung von Bedingungen für den korrekten Ablauf eines Programms, z.B. mittels Zusicherungen (*assertions*).

```
1  // naumann@stce.rwth-aachen.de
2
3  #include <iostream>
4  #include <cassert> // assertions for defensive programming
5
6  int main() {
7    float p=2;
8    float x=0;
9    std::cout << "x=";
10   std::cin >> x;
11   assert(0<=x&&x<=10); // assert valid x-range
12   float y=p*x;
13   std::cout << "y=" << y << std::endl;
14   return 0;
15 }
```

→ Live: Tour_MF002.cpp

Zusammenfassung

Modern Family Fallstudie

Erstes C++ Programm

Entwicklungsumgebung

Übersetzungsprozess

- GNU C++ Compiler `g++`
- Automatisierung mittels `make`

Defensive Programmierung