Übungsblatt 2 – Musterlösung:
C++ Libraries, Git & CMake

Sommersemester 2016

Programmiertechniken

Prof. Dr. L. Pollet

J. Greitemann, D. Hügel, J. Nespolo, T. Pfeffer

---

1) C++ Projekt mit mehreren Dateien

(a) `stat.cpp`:

```
#include ”stat.hpp”

double sum = 0.;
int N = 0;

void add(double x) {
   sum += x;
   ++N;
}

double mean() {
   return sum / N;
}
```

`stat.hpp`:

```
#pragma once

void add(double);
double mean();
```

`main.cpp`:

```
#include <iostream>
#include ”stat.hpp”

int main (int argc, char *argv[]) {
   double x;
   while (std::cin >> x) {
      add(x);
   }
   std::cout << ”Average temperature in Munich in 2015: ”
          << mean() << std::endl;
   return 0;
}
```

Kompilieren Sie mit `g++`:

```
g++ -o avg_temp main.cpp stat.cpp
./avg_temp
```

Geben Sie eine Reihe von Zahlen ein, separiert durch Druck auf die Return-Taste. Beenden Sie die Eingabe durch `Ctrl+D`.

(b) Testen Sie ihr Programm anhand der Tagesdurchschnittstemperaturen des Jahres 2015 in München, die Sie unter `https://db.tt/kET7HUlZ` herunterladen können. Die Jahresdurchschnittstemperatur sollte 11.1 ℃ betragen. Mit `./avg_temp < munich_2015.txt` können Sie die Daten direkt an das Programm übergeben.

(c) Legen Sie ein Git-Repository für dieses Projekt an und committen Sie den aktuellen Stand. Nur Quellcode, nicht aber Objektdateien oder ausführbare Programme, sollten von der Versionsverwaltung erfasst werden. Relative Pfade zu Dateien, die Git ignorieren soll, schreiben Sie in eine Datei `.gitignore`.

```
git init
git status
```

`.gitignore`:

```
avg_temp
munich_2015.txt
*.o
*.a
*.so
build
```

Das Sternchen ist eine sog. *Wildcard*, d.h. eine Art Platzhalter. Alle Dateien, die in `.o` enden werden mit `*.o` von `git` ignoriert.

```
git status
git add .
git status
git commit
git status
git log
```

(d) Nun wollen wir zusätzlich die Varianz berechnen und als Standardabweichung ausgeben. Legen Sie einen *Branch* an und wechseln Sie in diesen:

```
git branch
git branch variance
git status
git checkout variance
git status
```

Ergänzen Sie die folgenden Zeilen an geeigneter Stelle in den respektiven Dateien:

`stat.cpp`:

```
double sum_sq = 0.;
...
sum_sq += x*x;
...
double variance() {
   return (sum_sq - sum * sum / N) / (N-1);
}
```

`stat.hpp`:

```
...
double variance();
```

`main.cpp`:

```
#include <cmath>
...
   std::cout << ”Standard deviation: ” << sqrt(variance())
           << std::endl;
```

```
git status
git commit -a
git log
git checkout master
git log
```

2) Erstellen einer *static library* von Hand

Die Statistikfunktionalität des soeben entwickelten Programms wollen wir nun vom Anwendercode trennen und in eine statisch gelinkte Library auslagern.

(a) Verwenden Sie `ar` (und indirekt `ranlib`) um eine static library `libstat.a` zu erzeugen.

Kompilieren und Linken sind zwei separate Prozesse. Bisher hat `g++` beides gleichzeitig für uns übernommen. Wir können sie aber auch explizit trennen:

```
rm avg_temp
g++ -c stat.cpp
g++ -c main.o
g++ -o avg_temp main.o stat.o
ls
```

Mit `ar` kann eine Reihe von Objektdateien (`*.o`) zu einem Archiv kombiniert werden. Zusammen mit einem Index bildet ein solches Archiv eine Static Library.

```
rm avg_temp
ar rvs libstat.a stat.o
ls
```

Die Flag `s` von `ar` führt indirekt `ranlib` aus und erstellt den Index. Informieren Sie sich über die Funktion der übrigen Flag mit `man ar`.

(b) Kompilieren Sie nun den Anwendercode erneut und linken Sie diesen gegen die Library um das ausführbare Programm auf diesem Weg zu erstellen.

```
g++ -o avg_temp main.cpp -L. -lstat
./avg_temp < munich_2015.txt
```

(c) Löschen Sie ihre Library wieder. Ihr Programm funktioniert immer noch. Statisch gelinkte Bibliotheken werden in die ausführbare Datei eingebunden. Die Library wird nur zum Kompilieren benötigt. Wenn Sie ihr Programm vertreiben, müssen Sie Nutzer die Bibliothek nicht auf ihren Clients installieren.

```
rm libstat.a
./avg_temp < munich_2015.txt
```

Sorgen Sie dafür, dass am Ende dieser Aufgabe ihr Verzeichnis wieder sauber ist, d.h. keine von Git nicht erfassten Änderungen vorliegen:

```
rm stat.o main.o avg_temp
```

3) Verwenden von *CMake*; Git Merges

(a) `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 2.6)
project(avg_temp CXX)
add_executable(${PROJECT_NAME} main.cpp)
add_library(stat stat.cpp)
target_link_libraries(${PROJECT_NAME} stat)
```

```
mkdir build
cd build
cmake ..
make
./avg_temp < ../munich_2015.txt
```

Alternativ können Sie `make` mit `VERBOSE=1` aufrufen. `cmake` zeigt Ihnen dann, was es hinter den Kulissen tut. Finden Sie die Aufrufe von `ar`, `ranlib`, sowie dem Linker:

```
rm -rf *
cmake ..
make VERBOSE=1
```

(b) Erstellen Sie einen Git-Commit mit den in (a) vorgenommenen Änderungen auf dem `master` Branch.

```
git status
git add ../CMakeLists.txt
git commit
```

(c) Sie entscheiden sich nun, dass das in 1 (d) entwickelte Feature fertig ist.

```
git merge master variance
git log
```

`git` hat einen Merge Commit angelegt. Stattdessen können Sie auch einen Rebase von `variance` auf `master` durchführen. Informieren Sie sich hierüber in `man git-rebase`. Machen sie zunächst den Merge Commit rückgängig. Sie verlieren hierbei keinen Fortschritt, da der Branch `variance` noch intakt ist.

```
git reset --hard HEAD~1
git checkout variance
man git-rebase
```

Führen Sie jetzt den Rebase durch, gefolgt von einem abermaligen Merge. Da `variance` jetzt direkt auf den aktuellen `master` aufbaut, benötigt `git` keinen separaten Merge Commit und kann stattdessen einen sog. *Fast-forward* Merge durchführen. Den Branch `variance` benötigen Sie danach nicht mehr:

```
git rebase master variance
git checkout master
git merge variance
git branch -d variance
cmake ..
make
./avg_temp < ../munich_2015.txt
```

(d) Ändern Sie ihre `CMakeLists.txt` so ab, dass statt einer statisch gelinkten Library `libstat.a` eine dynamisch gelinkte `libstat.so` erzeugt wird.

`CMakeLists.txt`:

```
add_library(stat SHARED stat.cpp)
```

```
rm -rf *
cmake ..
make VERBOSE=1
ls
./avg_temp < ../munich_2015.txt
```

Wenn Sie jetzt die Shared Library löschen erhalten Sie zur Laufzeit einen Fehler. Das ausführbare Programm enthält nicht den Inhalt der Library, sondern linkt diesen zur Laufzeit (d.h. *dynamisch*).

```
rm libstat.so
./avg_temp < ../munich_2015.txt
make
git commit -a
```

(e) *(sehr fortgeschritten)* Gliedern Sie die Library komplett aus dem Anwendungsprojekt aus und machen Sie sie installierbar, d.h. fügen Sie `install`-Anweisungen zu `CMakeLists.txt` hinzu um die Library, den Header und ein Konfigurationsscript `stat-config.cmake` in die entsprechenden Installationsverzeichnisse zu kopieren. Verwenden Sie dann `find_package` im Anwendungsprojekt um die installierte Library ohne weiteres Dazutun des Users auf der Zielplattform zu finden.

```
cd ..
mkdir stat
git mv stat.* stat
ls *
cp CMakeLists.txt stat
cd stat
mkdir build
echo build > ./gitignore
```

`stat/statConfig.cmake.in`

```
add_library(stat SHARED IMPORTED)
find_library(STAT_LIBRARY_PATH stat
          HINTS "@CMAKE_INSTALL_PREFIX@/lib")
set_target_properties(stat PROPERTIES
                 IMPORTED_LOCATION "${STAT_LIBRARY_PATH}")
include_directories("@CMAKE_INSTALL_PREFIX@/include")
```

`stat/CMakeLists.txt`:

```
cmake_minimum_required(VERSION 2.6)
project(stat CXX)
add_library(stat SHARED stat.cpp)

configure_file (statConfig.cmake.in statConfig.cmake @ONLY)
install (FILES ${PROJECT_BINARY_DIR}/statConfig.cmake
       DESTINATION CMake)
install (FILES stat.hpp DESTINATION include)
install(TARGETS stat LIBRARY DESTINATION lib)
```

```
cd build
cmake ..
make
sudo make install
cd ../..
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.6)
project(avg_temp CXX)
add_executable(${PROJECT_NAME} main.cpp)
find_package(stat CONFIG REQUIRED)
target_link_libraries(${PROJECT_NAME} stat)
```

```
cd build
cmake ..
make
./avg_temp < ../munich
git add ..
git commit
```