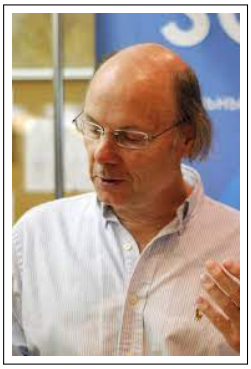
Tutorium Programmierkurs

Sommer 2022 **Übung 3**

Bjarne Stroustrup ist Professor der Informatik an der Texas A&M University. Bekanntheit erlangte er vor allem durch die Entwicklung der Programmiersprache C++, an deren Standardisierung er bis heute beteiligt ist. Aktuell ist er Gastprofessor an der Columbia University und arbeitet bei Morgan Stanley.
Stroustrup erwarb akademische Grade in Mathematik und Informatik (1975) an der Universität Aarhus sowie einen Doktorgrad in Informatik (1979) von der Universität Cambridge, England. Er leitete die Abteilung AT&T Lab's Large-scale Programming Research der Bell Laboratories von deren Gründung bis Ende 2002.
Seinen eigenen Worten zufolge hat Stroustrup „C++ erfunden, seine ersten Definitionen geschrieben und die erste Implementierung geschrieben[...], [die] Entwurfskriterien für C++ formuliert, alle wichtigen Einrichtungen entworfen und [er] war zuständig für das Durchsehen von Erweiterungsvorschlägen im C++-Standardisierungsgremium". Stroustrup schrieb auch das Fachbuch Die C++-Programmiersprache (englischer Originaltitel The C++ Programming Language). Der Text wurde mehrfach neu aufgelegt, um der fortschreitenden Entwicklung der Sprache und den Arbeiten des C++-Standardisierungskomitees gerecht zu werden.



Bjarne Stroustrup.

Aufgabe 4.1: Versionskontrolle mit git

git ist, wie in der Vorlesung vorgestellt, ein verteiltes Versionskontrollsystem. In dieser Übung geht es darum, sich ein wenig mit dem System vertraut zu machen.

(a) Melden Sie sich bei `https://edu.ziti.uni-heidelberg.de` an.

(b) Konfigurieren Sie git auf Ihrem Computer und legen Sie einen SSH-Key an.
Achtung:

- Ersetzen Sie Name und Email!
- Falls Sie schon einen SSH-Key haben brauchen Sie keinen anlegen.
- Der SSH-Key kann mit einem Passwort gesichert werden, dies ist optional!

```
# git config:
git config --global user.name "Hans Vader"
git config --global user.email "vader@death.star"
# ssh key creation
```

```
# default values are fine, but may overwrite existing keys with same name
ssh-keygen -t ed25519
# display Public ssh-key
cat ~/.ssh/id_ed25519.pub
```

(c) Fügen Sie den SSH-Public-Key ihrem Account bei edu.ziti.uni-heidelberg.de hinzu.

- Rufen Sie `https://edu.ziti.uni-heidelberg.de/-/profile/keys` auf.
- Geben Sie den Output des 'cat' Befehls in das große Textfeld ein.
- Klicken Sie auf 'Add Key'.

(d) Legen Sie ein Projekt für Ihre Gruppe an. Besuchen Sie `https://edu.ziti.uni-heidelberg.de/ws2021-ipk/exercise-template/` und folgen der Anleitung dort.
Denken Sie daran auch Ihren Tutor zum Projekt hinzuzufügen, damit dieser Ihre Abgaben bewerten kann!

(e) Kopieren Sie die Quellcodedateien der bisherigen Übungen in die Entsprechenden Ordner, fügen Sie diese zu git hinzu und erstellen einen Commit. Beim Erstellen des Commits müssen Sie eine Commit Message eingeben. Dies können Sie auf zwei verschiedene Arten machen:

- Sie schreiben einfach `git commit`. In diesem Fall öffnet sich im Terminal ein Texteditor mit einigen auskommentierten Zeilen (beginnend mit `"#"`). Schreiben Sie in die erste (leere) Zeile die Commit Message (diese kann auch mehrere Zeilen lang sein), speichern Sie die Datei und verlassen Sie den Editor. Meistens öffnet sich unter Linux der Editor `vim`, der etwas gewöhnungsbedürftig ist:
  1. Drücken Sie die Taste `"i"` (für "insert"), um den Eingabemodus zu aktivieren.
  2. Tippen Sie die commit message.
  3. Drücken Sie die Taste `"ESC"`, um den Editiermodus zu verlassen.
  4. Geben Sie folgendes ein, um die Datei zu speichern und den Editor zu schliessen: `":wq"` und drücken Sie Enter.
- Alternativ können Sie die commit message auch auf der Kommandozeile eingeben:

  ```
  git commit -m "hier steht Ihre commit message"
  ```

  Damit ist es allerdings schwierig, mehrzeilige Commit Messages zu schreiben.

Hinweis: Kompilierte Programme sollten normalerweise nicht eingecheckt werden, da man diese ja einfach durch erneutes Kompilieren wieder aus den Quellcodedateien erzeugen kann.

(f) Pushen Sie den Commit auf den Server.

(g) Während der Arbeit in den Übungen entstehen diverse Dateien, die nicht mit unter Versionskontrolle sollen, z.B. Sicherungsdateien, `.o`-Dateien und CMake-Buildverzeichnisse. Um diese Dateien zu *ignorieren*, gibt es in git den `gitignore`-Mechanismus (siehe `git help gitignore`). Erstellen Sie in Ihrem Arbeitsverzeichnis eine Datei `.gitignore` (Achtung, diese Datei ist versteckt, zum Anzeigen verwenden Sie `ls -a`) mit Dateinamen, die nicht zu git hinzugefügt werden sollen, z.B.

```
# alle Dateien, die auf .o enden
*.o
# eine bestimmte Datei, z.B. ein Executable
statistics
# Sicherungskopien, die auf ~ enden
*~
# das Unterverzeichnis build/ (von CMake)
build/
```

Hierbei werden Zeilen, die mit `#` beginnen, wieder als Kommentare ignoriert. Fügen Sie die Datei zu git hinzu, erstellen Sie einen Commit und pushen Sie diesen.

Von nun an können Sie git verwenden, um Dateien zwischen Ihrem Laptop und dem Rechner im Computer-Pool zu synchronisieren.

(h) Testen Sie, ob alle wichtigen Dateien auf dem Server angekommen sind (am einfachsten über das Webinterface). **Danach** löschen Sie das lokale Verzeichnis mit Ihren Übungen, indem Sie in das übergeordnete Verzeichnis wechseln und den Befehl `rm -rf ipk-exercises` ausführen, wobei Sie `ipk-exercises` durch den Namen ersetzen müssen, den Sie für das Verzeichnis verwendet haben.

Im Anschluss klonen Sie das Repository wieder vom Server und überprüfen, dass alle eingecheckten Dateien weiterhin da sind.

Hinweis: Auch in Zukunft ist es sinnvoll, Ihre Dateien zumindest am Ende der Übungsstunde zu committen und auf den Server zu pushen. Dann können Sie das Repository von zu Hause erneut klonen und an den Übungen weiterarbeiten. Es ist auch eine gute Idee, regelmäßiger Commits zu machen, z.B. immer dann, wenn man eine Teilaufgabe erfolgreich gelöst hat.

Weitere Hilfe zu git finden Sie unter anderem hier:

- `https://git-scm.com/doc` Ausführliche Dokumentation und einige Einführungsvideos.
- `https://git-scm.com/docs/everyday` Eine praktische Kurzübersicht wichtiger Befehle.
- `https://git-scm.com/docs/gittutorial` Das offizielle Tutorial.

Ansonsten ist Google Ihr Freund...

Aufgabe 4.2: std::vector

Im folgenden lernen Sie den wichtigsten Container der C++-Standardbibliothek kennen: `std::vector<T>`, eine indizierte Liste mit Einträgen vom Typ `T`. `T` kann hierbei ein

(fast) beliebiger Datentyp sein, z.B. `int` oder `double`. Einen `std::vector` können Sie auf verschiedene Weisen anlegen:

```
#include <vector> // vector in Ihrem Programm verfügbar machen

int main(int argc, char** argv)
{
  // Ein leerer vector für ganze Zahlen
  std::vector<int> v1;
  // Ein vector für ganze Zahlen mit 10 Einträgen
  std::vector<int> v2(10);
  // Ein vector mit den Einträgen 3,8,7,5,9,2 (benötigt compiler option -std=c++11)
  std::vector<int> v3 = {{ 3, 8, 7, 5, 9, 2 }};
}
```

Ein `vector` ist ein *Objekt* und hat sogenannte *member functions*, das sind spezielle Funktionen, die das Objekt verändern. Eine vollständige Referenz finden Sie auf der Website `cppreference.com`[1], die wichtigsten Methoden für diese Aufgabe sind:

```
std::vector<int> v = {{ 3, 8, 7, 5, 9, 2 }};
// Gibt die Anzahl der Einträge zurück
std::cout << v.size() << std::endl; // 6
// Verändert die Länge der Liste
v.resize(42);
```

Um auf einen Eintrag des Vektors zuzugreifen, schreiben Sie den Index des Eintrags in eckigen Klammern hinter den Variablennamen. **Die Nummerierung der Einträge beginnt bei 0, nicht bei 1**. Um einen Eintrag zu verändern, weisen Sie dem Eintrag einfach einen neuen Wert zu:

```
// Zugriff auf einzelne Einträge - Index ist 0-basiert!
std::cout << v[2] << std::endl; // 7
v[0] = v[0] * 2;
std::cout << v[0] << std::endl; // 6

```

Aufgaben:

(a) Legen Sie einen `vector<double>` mit jeder der oben beschriebenen Methoden an und geben Sie jeweils alle Einträge mit einer `for`-Schleife aus. Welchen Wert haben Einträge, für die Sie keinen expliziten Wert angegeben haben?

(b) Schreiben Sie eine Funktion, die den grössten und den kleinsten Wert in einem Vektor findet und als `std::pair` zurückgibt (erst den kleinsten, dann den größten Wert). Testen Sie die Funktion mit verschiedenen Vektoren.

(c) Schreiben Sie eine Funktion `std::vector<double> reversed(const std::vector<double>& v)` die einen Vektor mit Einträgen $x_0, x_1, \ldots, x_{n-1}$ als Parameter nimmt und einen neuen Vektor mit den Einträgen in umgekehrter Reihenfolge $x_{n-1}, x_{n-2}, \ldots, x_0$ zurückgibt.

[1] `http://en.cppreference.com/w/cpp/container/vector`

Testen Sie die Funktion mit verschiedenen Vektoren, insbesondere auch mit einem leerem.

(d) Schreiben Sie eine Funktion, die alle Einträge in einem `std::vector<double>` auf ganze Zahlen rundet und diese dann wieder im **selben** Vektor speichert. Zum Runden von Zahlen verwenden Sie folgendes:

```
#include <cmath>

int main()
{
  double x = 2.71;
  double x_rounded = std::round(x);
}
```

Testen Sie die Funktion mit verschiedenen Vektoren.

(e) Schreiben Sie eine Funktion, das die Reihenfolge der Einträge in einem Vektor umkehrt, aber das Ergebnis nun im **selben** Vektor speichert. Verwenden Sie zum Vertauschen einzelner Einträge die Funktion `std::swap(a,b)`. Lesen Sie auf `cppreference.com`[2] nach, was diese Funktion macht und welche `#include`-Anweisung Sie benötigen. Testen Sie die Funktion mit leeren Vektoren sowie mit welchen, die eine gerade bzw. eine ungerade Grösse haben.

[2] `http://en.cppreference.com/w/cpp/algorithm/swap`