

U1 1. Übung

- Allgemeines zum Übungsbetrieb
- Nachtrag zur Benutzerumgebung
- Versionsverwaltung mit Subversion / SP-Abgabesystem
- Ergänzungen zu C
 - ◆ Portable Programme
 - ◆ Gängige Compiler-Warnungen
- Anforderungen an abgegebene Lösungen
- Aufgabe 0: hello

U1-1 Allgemeines zum Übungsbetrieb

1 Anlaufstellen

- Forum: <https://fsi.informatik.uni-erlangen.de/forum/18>
 - ◆ inhaltliche Fragen zum Stoff oder den Aufgaben
 - ◆ allgemein alles, was auch für andere Teilnehmer interessant sein könnte

- Mailingliste: i4sp@informatik.uni-erlangen.de
 - ◆ geht an alle Übungsleiter
 - ◆ Angelegenheiten, die nur die eigene Person/Gruppe betreffen

- der eigene Übungsleiter
 - ◆ Fragen zur Korrektur
 - ◆ fälschlicherweise positiver Abschreibetest

2 Hinweise zu den Aufgaben

- teils einzeln, teils in Zweier-Teams (siehe Aufgabenstellung)
 - ◆ bei Teamarbeit müssen beide Partner in der **gleichen** Tafelübung sein
- Korrektur und Bewertung erfolgt durch den jeweiligen Tafelübungsleiter
 - ◆ korrigierte Ausdrucke werden in der Tafelübung ausgegeben
 - ◆ eigenes Ergebnis nach Login im WAFFEL einsehbar
- Übungspunkte können das Klausurergebnis verbessern (Bonuspunkte)
 - ◆ Abschreibtests
 - ◆ Vorstellen der eigenen Lösung vor der Übungsgruppe (Anwesenheitspflicht)
- Bearbeitungszeit ist in Werktagen (bei uns: Montag bis Freitag) angegeben
 - ◆ Bearbeitungszeit enthält Tag der Tafelübung
 - ◆ Feiertage sind nicht enthalten (1.11.)
 - ◆ Abgabetermin kann per Skript erfragt werden (siehe U1.15)

U1-2 Nachtrag zur Benutzerumgebung

- UNIX-Grundkenntnisse werden vorausgesetzt
- Info: UNIX-Einführung der FSI
<http://fsi.informatik.uni-erlangen.de/vorkurs/>
- Die Übungsleiter sind in der Rechnerübung bei Bedarf behilflich

1 Dokumentation aus 1. Hand: Manual-Pages

- Aufgeteilt in verschiedene *Sections*
 - (1) Kommandos
 - (2) Systemaufrufe
 - (3) Bibliotheksfunktionen
 - (4) Dateiformate (spezielle Datenstrukturen, etc.)
 - (5) Dateiformate (spezielle Datenstrukturen, etc.)
 - (6) verschiedene (z.B. Terminaltreiber, IP, ...)
 - (7) verschiedenes (z.B. Terminaltreiber, IP, ...)
- man-Pages werden normalerweise mit der Section zitiert: **printf(3)**
- Aufruf unter Linux:

```
man [section] Begriff
```

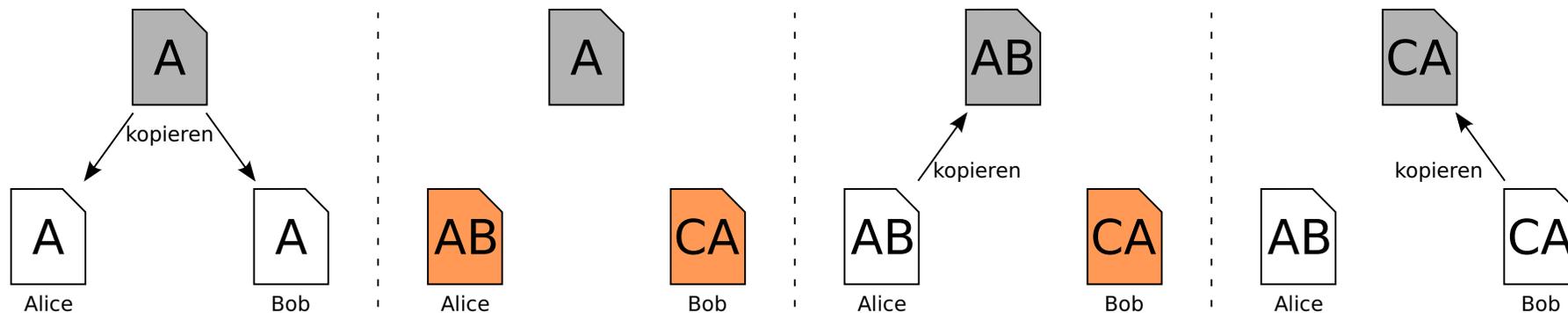
```
z.B. man 3 printf
```

- Suche nach Sections: **man -f Begriff**
Suche von man-Pages zu einem Stichwort: **man -k Stichwort**

U1-3 Versionsverwaltung

1 Warum Versionsverwaltung?

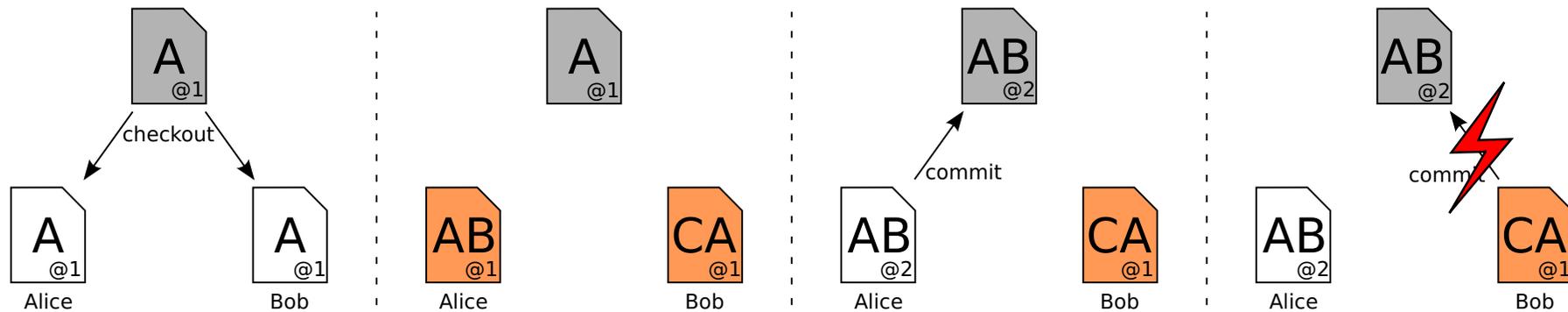
- Gemeinsames Bearbeiten einer Datei kann zu Problemen führen
- Beispiel: Gemeinsames Bearbeiten einer Datei ohne Versionsverwaltung



- ◆ Modifikationen werden nicht erkannt
- ◆ Änderungen von Alice gehen unbemerkt verloren

1 Warum Versionsverwaltung?

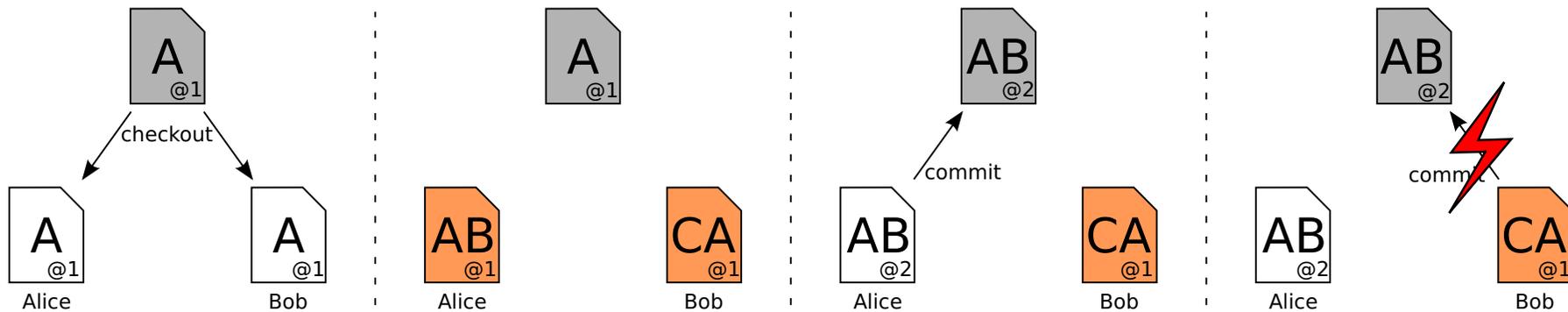
- Verwendung einer Versionsnummer zur Erkennung von Modifikationen



◆ Modifikationen werden erkannt

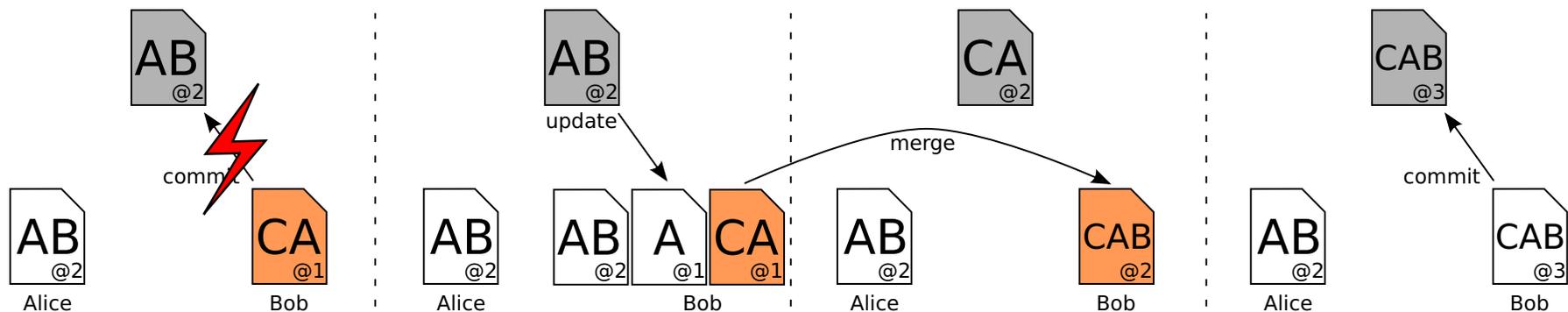
1 Warum Versionsverwaltung?

- Verwendung einer Versionsnummer zur Erkennung von Modifikationen



◆ Modifikationen werden erkannt

- Entstandener Konflikt muss lokal gelöst werden

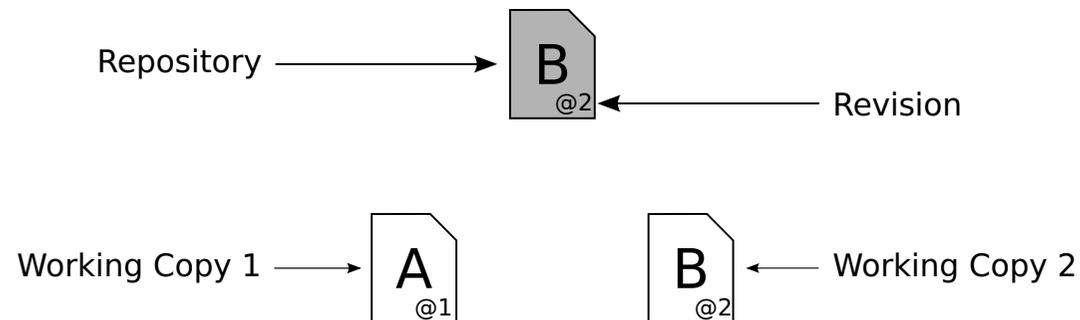


- Versionsverwaltung ermöglicht die gemeinsame Bearbeitung von Dateien

2 Das Versionsverwaltungssystem Subversion (SVN)

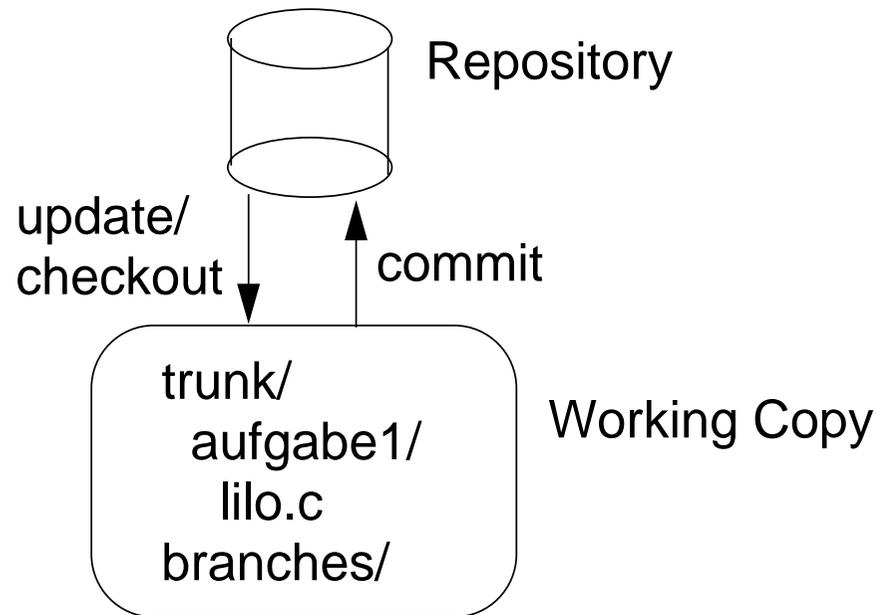
- SVN bietet Versionsverwaltung für Dateien und Verzeichnisse
- Archiviert Änderungen zentral in sogenanntem Repository
- Speichert Zusatzinformationen zu jeder Änderung:
 - ◆ Name des Ändernden
 - ◆ Zeitpunkt
 - ◆ Kommentar
- Kommando **svn**
- Grafische Frontends: TortoiseSVN (Windows), SCPlugin (Mac OS X)
- SP-Abgabesystem verwendet Subversion
- Ausführliche SVN-Dokumentation im Subversion-Buch
<http://svnbook.red-bean.com>

3 Terminologie bei Subversion



- Repository: zentrales Archiv aller Versionen
 - ◆ Zugriff erfolgt beispielsweise per Internet
- Revision (Versionsnummer)
 - ◆ Fortlaufend ab Revision 0 (1,2,3,...)
- Working Copy (Arbeitskopie)
 - ◆ lokale Kopie einer bestimmten Version des Repositories
 - ◆ kann versionierte und unversionierte Dateien und Verzeichnisse enthalten
 - ◆ es kann mehrere Arbeitskopien zu einem Repository geben (z.B. CIP/daheim)

4 Basisoperationen



- `checkout/co`: Anlegen einer neuen Arbeitskopie
- `update/up`: Neueste Revision vom Server holen
 - ◆ Bezieht sich auf aktuelles Verzeichnis und alle enthaltenen Verzeichnisse
- `commit/ci`: Einbringen einer neuen Version in das Repository ("Checkin")
 - ◆ Anmerkung: Umgebungsvariable `EDITOR` legt fest, welcher Editor zur Eingabe des Kommentars gestartet wird.

4 Basisoperationen

- add: Dateien unter Versionskontrolle stellen
 - ◆ Bei einer leeren Arbeitskopie müssen entsprechende Dateien oder Verzeichnisse erst eingefügt werden
- del/remove/rm: Datei lokal löschen und nicht länger unter Versionskontrolle halten
- status/st: Änderungen der Arbeitskopie anzeigen

```
$ svn status
A  aufgabe1/lilo.txt
M  aufgabe1/lilo.c
?  aufgabe1/lilo
```

- ◆ **A**: Datei wurde unter Versionkontrolle gestellt
- ◆ **M**: Dateiinhalt wurde verändert
- ◆ **?**: Datei steht nicht unter Versionskontrolle
- ◆ **!**: Datei steht unter Versionskontrolle, aber nicht mehr in der Arbeitskopie vorhanden

5 SP-Abgabesystem

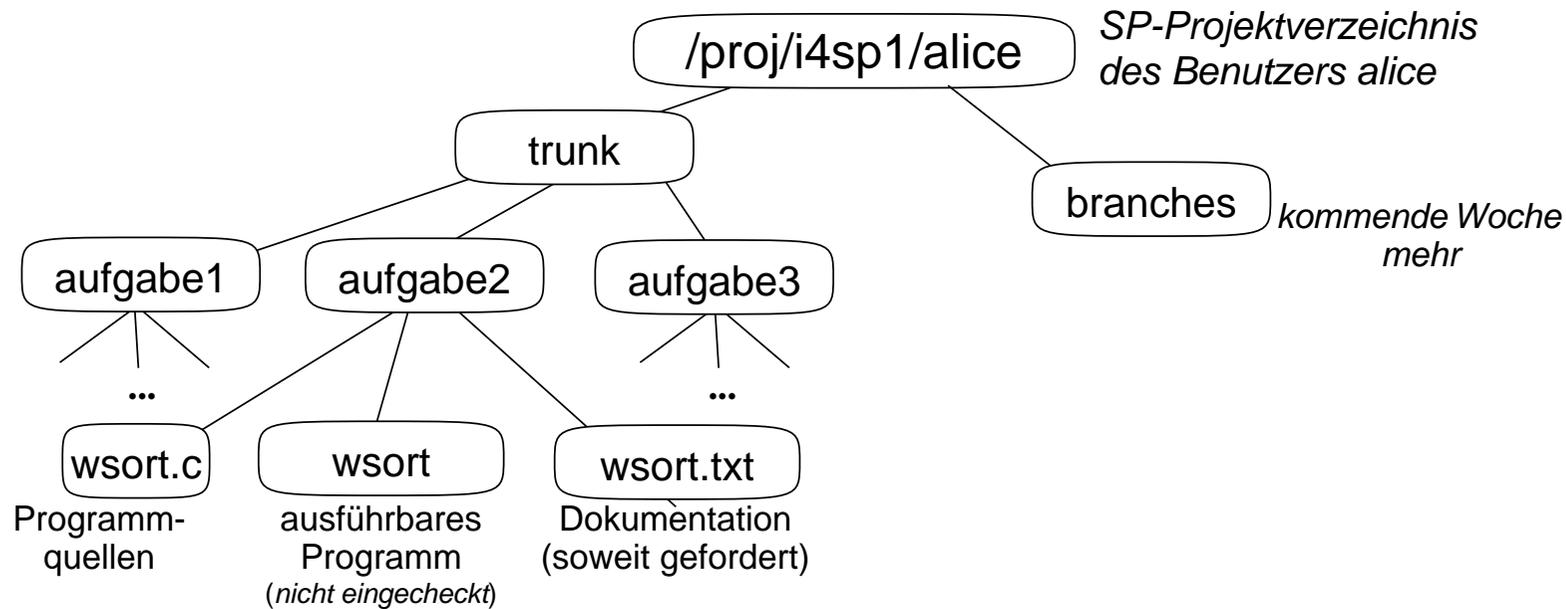
- Für jeden Teilnehmer wird nach der Anmeldung ein Repository erzeugt
<https://www4.informatik.uni-erlangen.de/i4sp/ws11/sp1/alice>
- Die Erzeugung erfolgt in der Nacht nach der Waffel-Anmeldung
- Im Projektverzeichnis wird eine Arbeitskopie des Repositories abgelegt
[/proj/i4sp1/alice](#)
- Zum Zugriff muss jeder Teilnehmer ein Subversion-Passwort setzen

```
$ /proj/i4sp1/bin/change-password
```

- Die Passwörter werden innerhalb der nächsten Stunde aktiv
- Sie können bei Bedarf weitere Arbeitskopien erzeugen (z.B. daheim)

```
$ svn co https://www4.informatik.uni-erlangen.de/i4sp/ws11/sp1/alice
```

6 Aufbau des SP-Repositorys



- Der *trunk* enthält ein Unterverzeichnis `aufgabe x` für jede Aufgabe
- Zur Abgabe folgendes Skript aufrufen

```
$ /proj/i4sp1/bin/submit aufgabe0
```

- ◆ dieses gibt die aktuellste Version Ihres Repositories ab
- ◆ offene Änderungen vor der Abgabe einchecken
- ◆ unterhalb von `branches` **nichts** von Hand editieren/einchecken

7 Abgabemodalitäten

- mehrmalige Abgabe ist möglich
 - ◆ durch erneuten Aufruf des *submit*-Skripts
- gewertet wird die letzte rechtzeitige Abgabe
 - ◆ Abgabetermin kann per Skript erfragt werden (*Im Beispiel: alice ist in T01*)

```
$ /proj/i4sp1/bin/get-deadline aufgabe0  
Abgabezeitpunkt fuer Aufgabe 1: lilo: 2011-10-26 17:30:00
```

- Abgaben nach dem Abgabezeitpunkt sind möglich
 - ◆ bei Vorliegen eines triftigen Grundes
 - ◆ Wertung nur nach expliziter Rücksprache mit dem Übungsleiter
 - ◆ ansonsten wird letzte rechtzeitige Abgabe gewertet
- Die Hilfsskripte sind nur im CIP-Pool verfügbar

8 Beispiel-Workflow für Aufgabe 1

```
alice@fau06a[~] cd /proj/i4sp1/alice/trunk
alice@fau06a[trunk] mkdir aufgabe0
alice@fau06a[trunk] cd aufgabe0
alice@fau06a[aufgabe0] vim hello.c
...
alice@fau06a[aufgabe0] cd ..
alice@fau06a[trunk] svn add aufgabe0
A   aufgabe0
A   aufgabe0/hello.c
alice@fau06a[trunk] svn commit
...
Committed revision 2.
alice@fau06a[trunk] vim aufgabe0/hello.c
...
alice@fau06a[trunk] svn commit -m 'Bugfix in printf'
...
Committed revision 3.
alice@fau06a[trunk] /proj/i4sp1/bin/submit aufgabe0
...
# Aufgabe 0 ist jetzt abgegeben
```

U1-4 Portable Programme

- Entwicklung portabler Programme durch Verwendung definierter Schnittstellen

1 ANSI-C

- Normierung des Sprachumfangs der Programmiersprache C
- Standard-Bibliotheksfunktionen (z. B. printf, malloc, ...)

2 Single UNIX Specification V3 (SUSv3)

- Standardisierung der Betriebssystemschnittstelle
- SUSv3 wird von verschiedenen Betriebssystemen implementiert:
 - ◆ SUN Solaris, HP/UX, AIX
 - ◆ Linux
 - ◆ Mac OS X (Darwin)

U1-5 Anforderungen an abgegebene Lösungen

- C-Sprachumfang konform zu ANSI-C99
- Betriebssystemschnittstelle konform zu SUSv3
- **warnings-** und **fehlerfrei** mit folgendem Aufruf übersetzen (Bsp. `hello`):

```
gcc -std=c99 -pedantic -D_XOPEN_SOURCE=600 -Wall -Werror -o hello hello.c
```

- ◆ `-std=c99 -pedantic` erlauben nur ANSI-C99 konformen C-Quellcode
- ◆ `-D_XOPEN_SOURCE=600` erlaubt nur SUSv3 konforme Betriebssystemaufrufe
- ◆ mit `-wall` werden weitere Warnungen aktiviert, die auf mögliche Programmierfehler hinweisen
- ◆ mit `-werror` werden alle Warnungen wie Fehler behandelt
- ◆ einzelne Aufgaben können hiervon abweichen, dies wird in der Aufgabenstellung entsprechend vermerkt

U1-6 Gängige Compiler-Warnungen

■ *implicit declaration of function 'printf'*

◆ bei Bibliotheksfunktion fehlt entsprechendes `#include`

- Entsprechende Manual-Page gibt Auskunft über den Namen der nötigen Headerdateien

```
$ man 3 printf
```

SYNOPSIS

```
#include <stdio.h>
```

```
int printf(const char *format, ...);
```

◆ bei einer eigenen Funktion fehlt die Forward-Deklaration

■ *control reaches end of non-void function*

◆ in einer Funktion, die einen Wert zurückliefern soll, fehlt an einem Austrittspfad eine passende `return`-Anweisung

U1-7 Aufgabe 0: hello - Formatierte Ausgabe

■ Bibliotheksfunktion — Prototypen (Schnittstelle)

```
int printf(const char *format, /* Parameter */ ...);
```

◆ Ausgabe erscheint (normalerweise) auf dem Bildschirm

■ Die statt ... angegebenen Parameter werden entsprechend der Angaben im **format**-String ausgegeben

◆ normale Zeichen: werden einfach auf die Ausgabe kopiert

◆ Escape-Zeichen: z. B. `\n` oder `\t`, werden durch die entsprechenden Zeichen (hier Zeilenvorschub bzw. Tabulator) bei der Ausgabe ersetzt

◆ Format-Anweisungen: beginnen mit %-Zeichen und beschreiben, wie der dazugehörige Parameter in der Liste nach dem **format**-String aufbereitet werden soll

■ Beispiel: Die Zeichenkette `Hallo Welt!` in einer eigenen Zeile ausgeben

```
printf("Hallo Welt!\n");
```