

U1 1. Übung

- Allgemeines zum Übungsbetrieb
- Nachtrag zur Benutzerumgebung
- Ergänzungen zu C
 - ◆ Portable Programme
 - ◆ Gängige Compiler-Warnungen
 - ◆ Dynamische Speicherverwaltung
- Versionsverwaltung mit Subversion / SP-Abgabesystem
- Anforderungen an abgegebene Lösungen
- Aufgabe 1: lilo

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 Ausdrücke 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 (2.6., 13.6., 14.6. und 23.6.)
 - ◆ Abgabetermin kann per Skript erfragt werden (siehe U1.18)

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
 - (5) Dateiformate (spezielle Datenstrukturen, etc.)
 - (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 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)

U1-4 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

U1-5 Dynamische Speicherverwaltung

- Dynamische Speicherverwaltung notwendig, wenn
 - ◆ Anzahl und/oder Größe der gespeicherten Daten erst zur Laufzeit bekannt ist
 - ◆ **und** der angeforderte Speicher keine strikt beschränkte Lebensdauer hat
- Beispiel: Erzeugen von Feldern der Länge **n** mittels `malloc()`

```
int *numbers;  
numbers = (int *)malloc(sizeof(int)*n);  
if(numbers == NULL) ...
```

- ◆ Zurückgebener Speicherbereich kann zufällige Werte enthalten

- Freigeben von Speicher

```
void free(void *ptr);
```

- ◆ nur Speicher, der mit einer der alloc-Funktionen zuvor angefordert wurde, darf mit free freigegeben werden!

U1-6 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

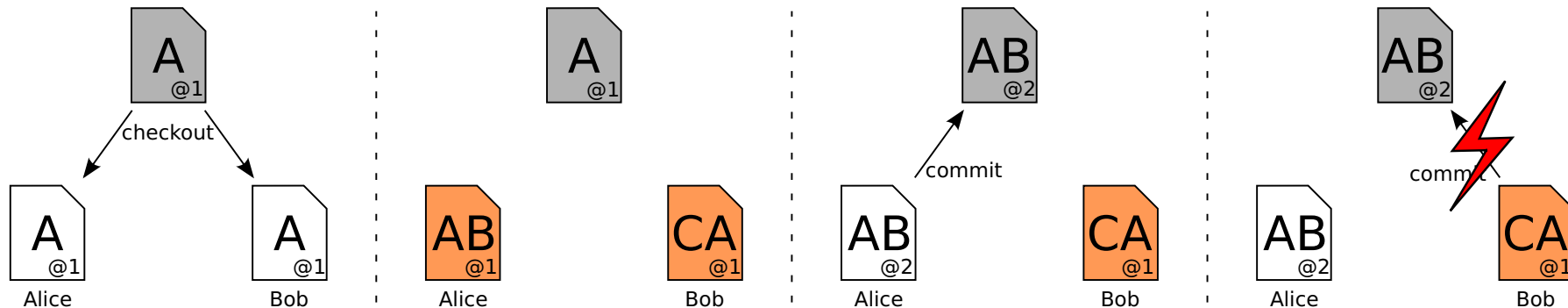
1 Warum Versionsverwaltung?

- Verwendung einer Versionsnummer zur Erkennung von Modifikationen

◆ Modifikationen werden erkannt

1 Warum Versionsverwaltung?

- Verwendung einer Versionsnummer zur Erkennung von Modifikationen



◆ Modifikationen werden erkannt

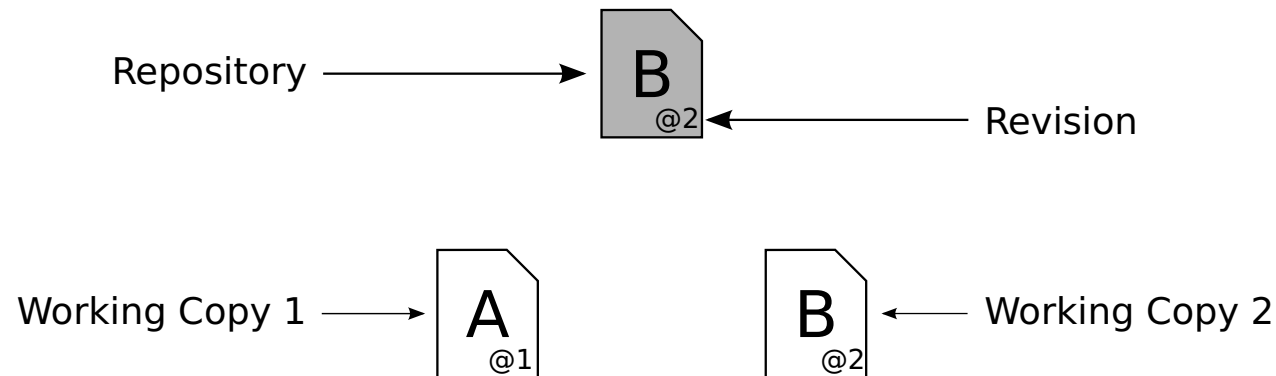
- Entstandener Konflikt wird lokal gelöst

- Versionsverwaltung ermöglicht die gemeinsame Bearbeitung von Dateien

2 Das Versionsverwaltungssystem Subversion (SVN)

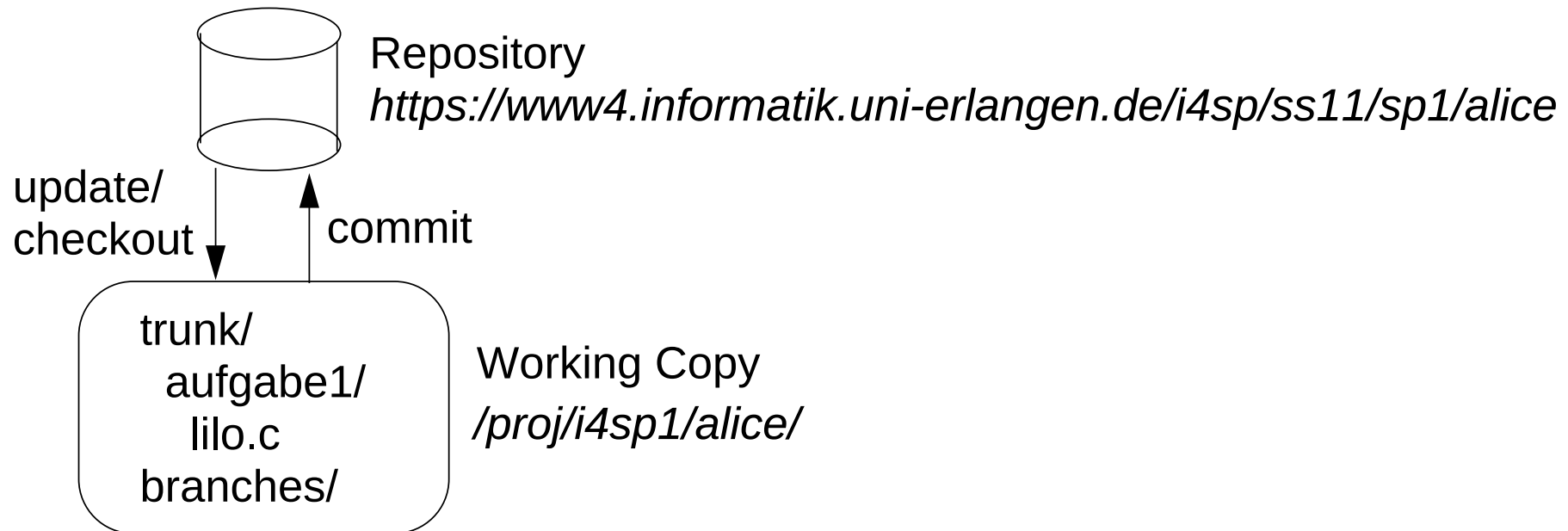
- Versionsverwaltung für Dateien und Verzeichnisse
- Archiviert Änderungen zentral in sogenanntem Repository
- Speichert Zusatzinformationen zu jeder Änderung:
 - ◆ Name des Ändernden
 - ◆ Zeitpunkt
 - ◆ Kommentar
- Kommando **svn**
- Grafische Frontends: TortoiseSVN (Windows), SCPlugin (Mac OS X)
- 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
(Bei uns: <https://www4.informatik.uni-erlangen.de/i4sp/ss11/sp1/<login>>)
- 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 löschen
- status/st: Änderungen in der Arbeitskopie anzeigen

```
alice@fau01[/proj/i4sp1/alice/trunk]$ 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

5 SP-Abgabesystem

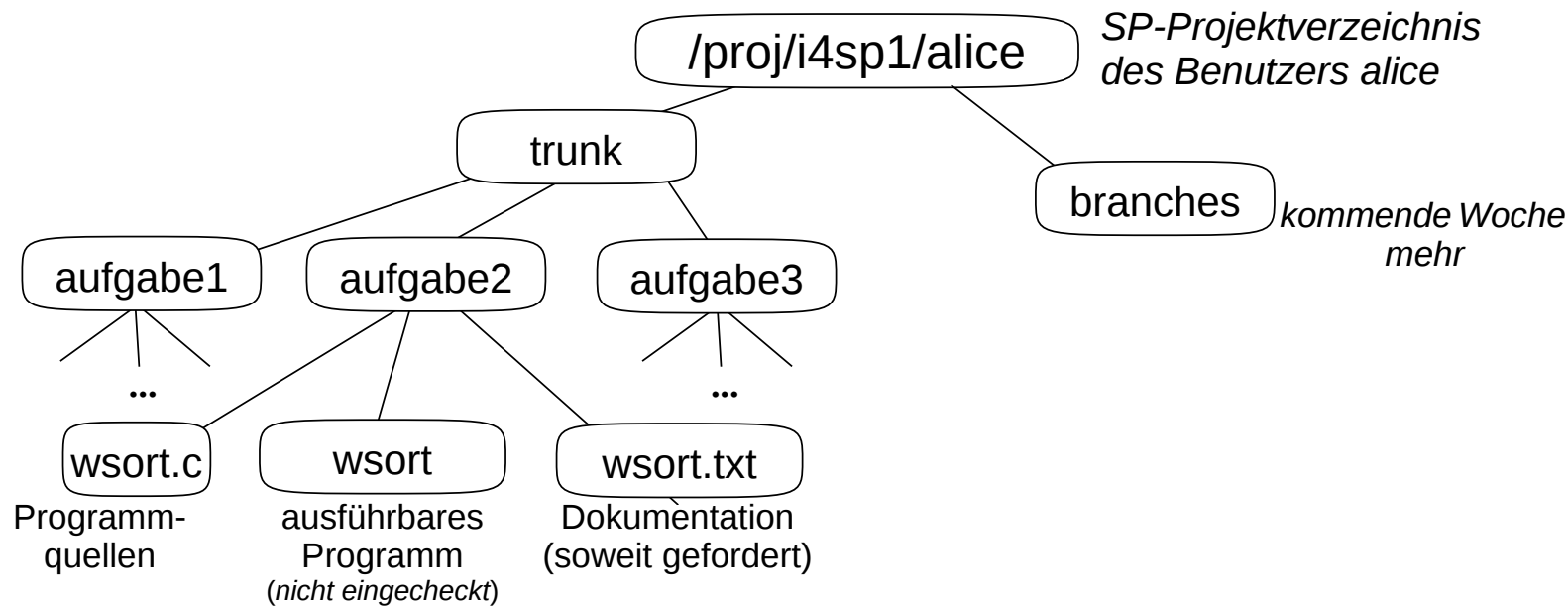
- Für jeden Teilnehmer wird nach Anmeldung ein Repository erzeugt
<https://www4.informatik.uni-erlangen.de/i4sp/ss11/sp1/alice>
- Die Erzeugung erfolgt in der Nacht nach der Waffel-Anmeldung
- Im Projektverzeichnis wird eine Arbeitskopie des Repositories abgelegt
[/proj/i4sp1/alice](#)
- Zum Zugriff muss jeder Teilnehmer ein Subversion-Passwort setzen

```
alice@fau101[alice]$ /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/ss11/sp1/alice
```

6 Aufbau des SP-Repositories



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

```
alice@fau101[aufgabe1]$ /proj/i4sp1/bin/submit aufgabe1
```

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

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*)

```
alice@fau01[aufgabe1]$ /proj/i4sp1/bin/get-deadline aufgabe1
Abgabezeitpunkt fuer Aufgabe 1: lilo: 2011-05-12 18:00:00
```

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

8 Beispiel-Workflow für Aufgabe 1

```
alice@fau01[~] cd /proj/i4sp1/alice/trunk
alice@fau01[trunk] mkdir aufgabe1
alice@fau01[trunk] cd aufgabe1
alice@fau01[aufgabe1] vim lilo.c
...
alice@fau01[aufgabe1] cd ..
alice@fau01[trunk] svn add aufgabe1
A   aufgabe1
A   aufgabe1/lilo.c
alice@fau01[trunk] svn commit
...
Committed revision 2.
alice@fau01[trunk] vim aufgabe1/lilo.c
...
alice@fau01[trunk] svn commit -m 'Bugfix in insertElement'
...
Committed revision 3.
alice@fau01[trunk] /proj/i4sp1/bin/submit aufgabe1
...
# Aufgabe 1 ist jetzt abgegeben
```

U1-7 Anforderungen an abgegebene Lösungen

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

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

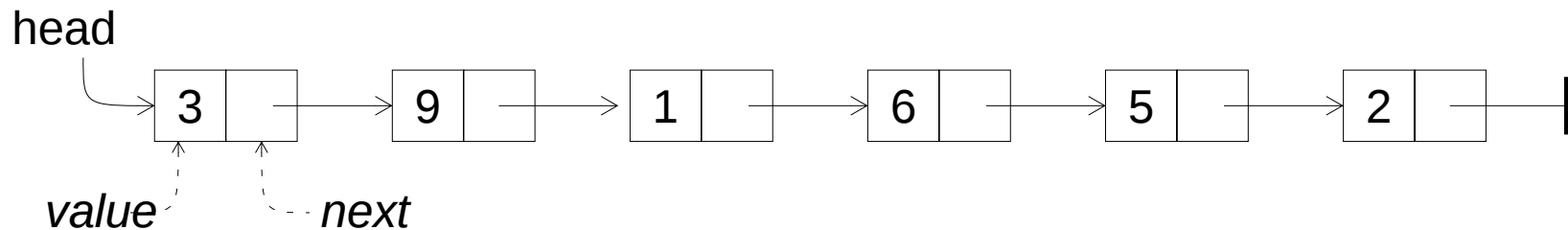
- ◆ `-std=c99 -pedantic` erlauben nur ANSI-C99 konformen C-Quellcode
- ◆ `-D_XOPEN_SOURCE=600` erlaubt nur SUSv3 konforme Betriebssystemaufrufe
- ◆ mit `-Wall` werden weitere Warnungen aktiviert, die auf mögliche Programmierfehler hinweisen
- ◆ mit `-Werror` werden alle Warnungen wie Fehler behandelt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

U1-8 Aufgabe 1: lilo (last in - last out)

1 Einfach verkettete FIFO-Liste

■ Zielsetzungen

- ◆ Kennenlernen der Umgebung und Entwicklungswerkzeuge
- ◆ Dynamische Speicherverwaltung und Umgang mit Zeigern
- ◆ Verwendung des Abgabesystems



■ Strukturdefinition:

```

struct listelement {
    int value;
    struct listelement *next;
};
typedef struct listelement listelement; // optional
  
```

2 Schnittstelle

- **int insertElement(int value):** Fügt einen neuen, nicht-negativen Wert in die Liste ein, wenn dieser noch nicht vorhanden ist. Rückgabe *value* im Erfolgsfall, sonst -1.
- **int removeElement():** Entfernt den ältesten Wert in der Liste und gibt diesen zurück. Ist die Liste leer, wird -1 zurückgeliefert.

3 Dynamische Speicherverwaltung

- Die benötigte maximale Menge an Listenelementen ist nicht bekannt
- Listenelemente müssen bei Bedarf dynamisch allokiert werden
- Listenelemente freigeben, wenn diese nicht mehr benötigt werden