U1 1. Übung

- Allgemeines zum Übungsbetrieb
- Nachtrag zur Benutzerumgebung
- Ergänzungen zu C
  - Portable Programme
  - Gängige Compiler-Warnungen
  - Dynamische Speicherverwaltung
- Versionsverwaltung mit Subversion / SP-Abgabesystem
- 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 Abschreibtest

2 Bearbeitung und Abgabe der Aufgaben

- teils einzeln, teils in Zweier-Teams (siehe Aufgabenstellung)
- bei Teamarbeit müssen beide Partner in der gleichen Tafelübung sein
- Korrektur und Bewertung durch den jeweiligen Tafelübungsleiter
  - korrigierte Ausdrucke werden in der Tafelübung ausgegeben
  - eigenes Ergebnis außerdem nach Login im WAFFEL einsehbar
- Übungspunkte können das Klausurergebnis verbessern (Bonuspunkte)
  - Abschreibtests
  - Vorstellen der eigenen Lösung vor der Übungsgruppe
  - Anwesenheitspflicht

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 Quoting von Zeichen mit Sonderbedeutung

- Sonderzeichen (wie **<**, **>**, **&**, Space) soll als Argument übergeben werden:

```
cd Eigene Dateien
```

- Problem: die Shell interpretiert diese Zeichen
  - im Beispiel das Leerzeichen als Trenner mehrerer Argumente
  - das Kommando wird mit zwei Argumenten ausgeführt: *Eigene* und *Dateien*
- Lösung: Quoting nimmt Zeichen die Sonderbedeutung:
  - Voranstellen von \ nimmt genau einem Zeichen die Sonderbedeutung
    \ selbst wird durch \\ eingegeben
  - Klammern des gesamten Arguments durch " ",
    " selbst wird durch **\"** angegeben
  - Klammern des gesamten Arguments durch ’ ’,
    ’ selbst wird durch **\’** angegeben

```
cd "Eigene Dateien"; cd ’Eigene Dateien’; cd Eigene\ Dateien
```


2 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

(1) Verwenden eines verbreiteten Programmiersprachenstandards
Hier: ANSI-C99

(2) Verwenden einer standardisierten Betriebssystemschnittstelle
Hier: Single UNIX Specification V3 (SUSv3)
http://www.unix.org/single_unix_specification/

1 ANSI-C

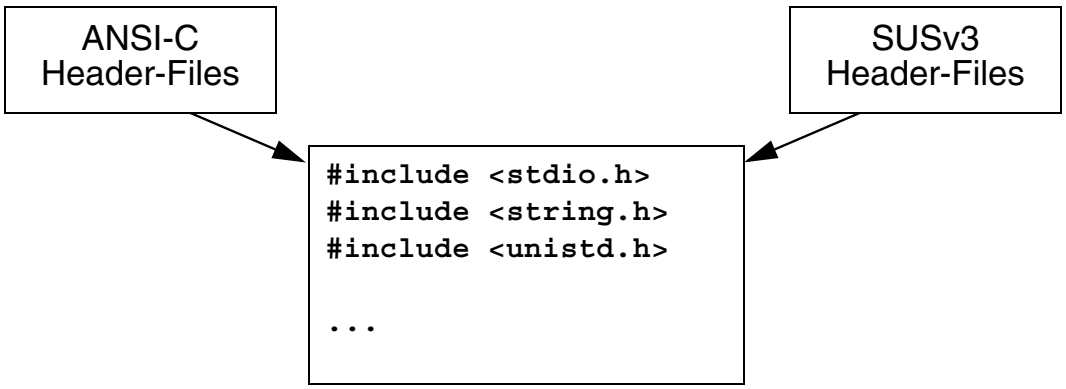
- Normierung des Sprachumfangs der Programmiersprache C
- Standard-Bibliotheksfunktionen
(z. B. printf, malloc, …)
- GCC-Aufrufoptionen: `-std=c99 -pedantic`

2 Single UNIX Specification

- Standardisierung der Betriebssystemschnittstelle
- SUSv3 wird von verschiedenen Betriebssystemen implementiert:
  - SUN Solaris, HP/UX, AIX
  - Linux
  - Mac OS X (Darwin)
- GCC-Aufrufoption: `-D_XOPEN_SOURCE=600`

3 Header-Files: ANSI-C und SUSv3

- In den Standards ANSI-C und SUSv3 sind Header-Files definiert, mit
  - Funktionsdeklarationen (auch Funktionsprototypen genannt)
  - typedefs
  - Makros und defines
  - Wenn in der Aufgabenstellung nicht anders angegeben, sollen ausschließlich diese Header-Files verwendet werden.



4 Betriebssystemabhängige Datentypen

- Typ-Deklarationen über typedef-Anweisung — Beispiel

```
typedef unsigned long dev_t;
dev_t device;
```

- Betriebssystemabhängige Typen aus `<sys/types.h>`:
  - `dev_t`: Gerätenummer
  - `gid_t`: Gruppen-ID
  - `ino_t`: Seriennummer von Dateien (Inodenummer)
  - `mode_t`: Dateiattribute (Typ, Zugriffsrechte)
  - `nlink_t`: Hardlink-Zähler
  - `off_t`: Dateigrößen
  - `pid_t`: Prozess-ID
  - `size_t`: entspricht dem ANSI-C size_t
  - `ssize_t`: Anzahl von Bytes oder -1
  - `uid_t`: User-ID

5 Anforderungen an abgegebene Lösungen

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

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

- 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

U1-4 Gängige Compiler-Warnungen

- *implicit declaration of function 'printf'*
  - handelt es sich um eine Bibliotheksfunktion, wurde ein `#include` vergessen (im Falle von printf: `#include <stdio.h>`)
  - 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

- Erzeugen von Feldern der Länge n:
  - mittels: `void *malloc(size_t size)`

```
struct person *personen;
personen = (struct person *)malloc(sizeof(struct person)*n);
if(personen == NULL) ...
```

  - mittels: `void *calloc(size_t nelem, size_t elsize)`

```
struct person *personen;
personen = (struct person *)calloc(n, sizeof(struct person));
if(personen == NULL) ...
```

  - `calloc` initialisiert den Speicher mit `0`
  - `malloc` initialisiert den Speicher nicht
  - explizite Initialisierung mit `void *memset(void *s, int c, size_t n)`

```
memset(personen, 0, sizeof(struct person)*n);
```


U1-5 Dynamische Speicherverwaltung

- Verändern der Größe von Feldern, die durch malloc bzw. calloc erzeugt wurden:

```
void *realloc(void *ptr, size_t size);
```

```
neu = (struct person *)realloc(personen,
                               (n+10) * sizeof(struct person));
if(neu == NULL) ...
```

- Freigeben von Speicher

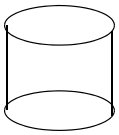
```
void free(void *ptr);
```

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

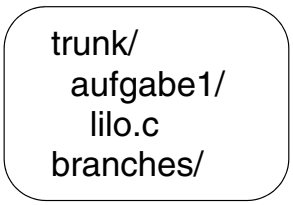
U1-6 Versionsverwaltung mit Subversion

- Versionsverwaltung für Dateien und Verzeichnisse
- Archiviert Änderungen in zentralem Repository
  - Name des Ändernden
  - Zeitpunkt
  - Kommentar
- Koordiniert Zugriffe verschiedener Benutzer
- Verwendet als Abgabesystem in den SP-Übungen
- Kommando `svn`
- Grafische Frontends
  - TortoiseSVN (Windows)
  - SCPlugin (Mac OS X)

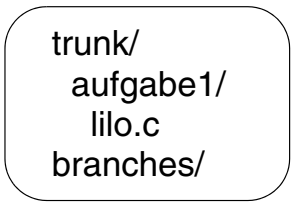
1 Repository und Working Copies



Repository
https://www4.informatik.uni-erlangen.de/i4sp/ws10/alice



Working Copy
/proj/i4sp/alice/
CIP-Pool



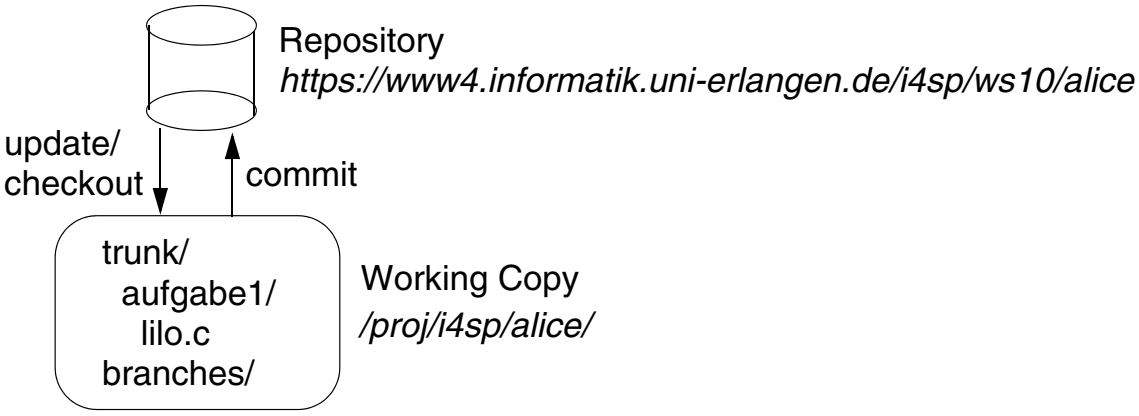
Working Copy 2
/home/alice/sp
Zuhause

- Repository: Archiv aller Versionen, zentral gespeichert
- Working Copy (Arbeitskopie)
  - lokale Kopie einer bestimmten Version des Repositories
  - Änderungen finden in der Arbeitskopie statt
  - es kann mehrere Arbeitskopien zu einem Repository geben (CIP/daheim)

2 Versionierungsschema

- Subversion nummeriert fortlaufend ab Revision 0 (1,2,3,...)
- spezielle Revisionsschlüsselwörter
  - HEAD: aktuelle Version des Repositories (neuste Version)
  - BASE: Revision eines Eintrags (Datei, Verzeichnis) der Arbeitskopie
  - COMMITTED: Letzte Änderungsrevision eines Eintrags älter als BASE
  - PREV: COMMITTED-1
- Revision zu einem bestimmten Zeitpunkt
  - {"2010-10-19 14:42"}

3 Basisoperationen



- checkout/co: Anlegen einer neuen Arbeitskopie
- update/up: Änderung der Revision einer Arbeitskopie

```
alice@faui01[/proj/i4sp/alice]$ svn update
U   trunk/aufgabe1/lilo.c
```

- commit/ci: Einbringen einer neuen Version in das Repository ("Checkin")

3 Basisoperationen

- log: Historie anzeigen

```
alice@faui01[/proj/i4sp/alice]$ svn log
------------------------------------------------------------------------
r1 | www-data | 2010-04-20 15:03:14 +0200 (Tue, 20 Apr 2010) | 1 line

init repository
------------------------------------------------------------------------
```

- status/st: Änderungen in der Arbeitskopie anzeigen

```
alice@faui01[/proj/i4sp/alice/trunk]$ svn status
A   aufgabe1
A   aufgabe1/lilo.c
```

- diff: Änderungen zwischen zwei Revisionen anzeigen

```
alice@faui01[/proj/i4sp/alice/trunk]$ svn diff
... diff mit Änderungen an der Arbeitskopie ...
```


3 Basisoperationen

- add: Dateien unter Versionskontrolle stellen

```
alice@faui01[/proj/i4sp/alice/trunk]$ svn add aufgabe1
A   aufgabe1
A   aufgabe1/lilo.c
```

- move/mv: Datei umbenennen oder verschieben
- del/remove/rm: Datei löschen
- revert: Änderungen an Arbeitskopie zurücksetzen

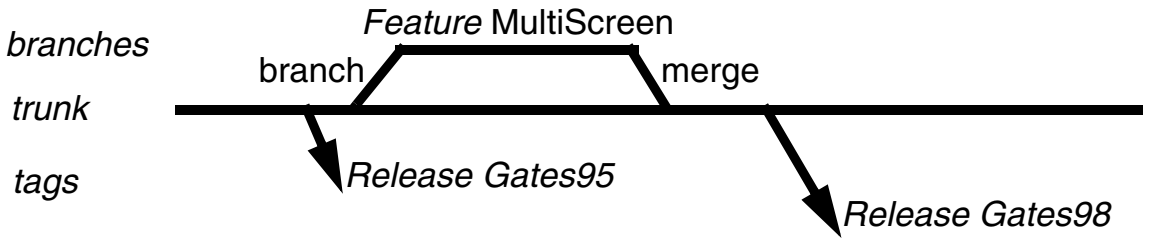
```
alice@faui01[/proj/i4sp/alice/trunk]$ svn revert aufgabe1
Reverted 'aufgabe1'
# svn add aus obigem Beispiel wurde rückgängig gemacht
```

- copy/cp: Datei/Teilbaum kopieren

```
alice@faui01[trunk]$ svn cp aufgabe1 ../branches/aufgabe1
# aufgabe1 in trunk wurde in branches kopiert
```

4 Konventionelles Repository-Layout

SVN-Repository des Projekts *Gates*



- Unterteilung des Wurzelverzeichnisses
  - Hauptentwicklungslinie: *trunk*
  - Verzeichnis mit Entwicklungszweigen: *branches*
  - Eingefrorene Versionen: *tags*
- Größere Features können von der Hauptlinie entkoppelt in einem eigenen Zweig (*branch*) entwickelt werden und nach Fertigstellung wieder in die Hauptlinie eingebracht (*merge*) werden
- Besondere Versionen können benamt (*getagged*) werden (z.B. Release)

5 SP-Abgabesystem

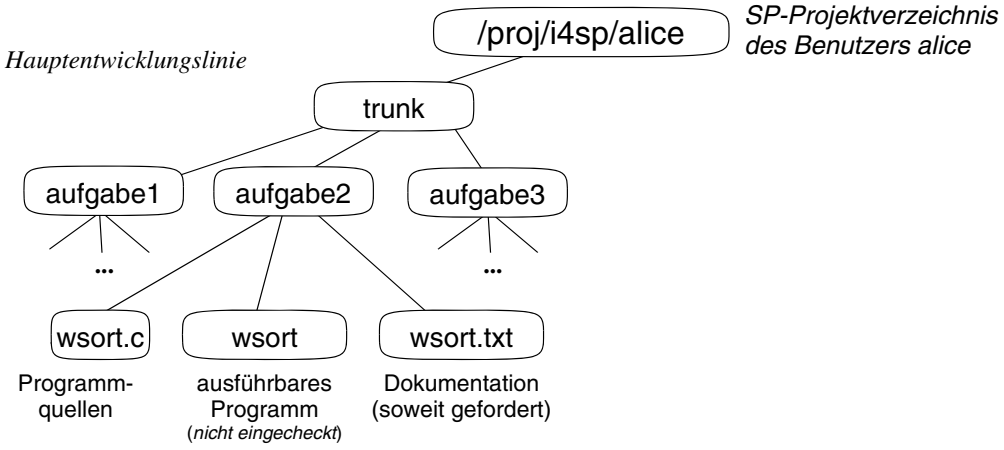
- Für jeden Teilnehmer wird nach Anmeldung ein Repository erzeugt *https://www4.informatik.uni-erlangen.de/i4sp/ws10/alice*
- Die Erzeugung erfolgt in der Nacht nach der Waffel-Anmeldung
- Im Projektverzeichnis wird eine Arbeitskopie des Repositories abgelegt */proj/i4sp/alice*
- Zum Zugriff muss jeder Teilnehmer sein Subversion-Passwort setzen

```
alice@faui01[alice]$ /proj/i4sp/bin/change-password
```

- Die Passwörter werden zur nächsten vollen Stunde aktiv
- Sie können bei Bedarf weitere Arbeitskopien erzeugen (z.B. zuhause)

```
$ svn co https://www4.informatik.uni-erlangen.de/i4sp/ws10/alice
```

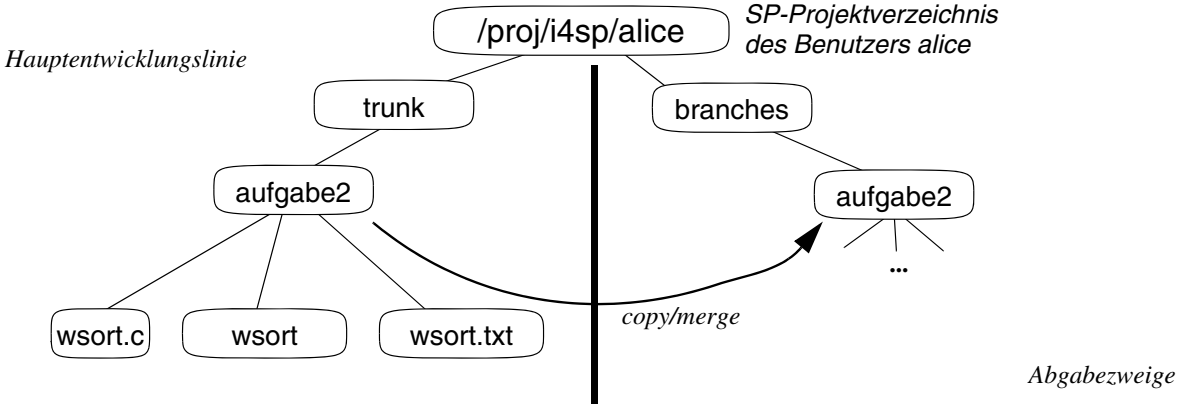
6 Aufbau des SP-Repositories



- Der *trunk* enthält ein Unterverzeichnis `aufgabe`**x** für jede Aufgabe
  - Sie können beliebige Zwischenstände in den *trunk* einchecken

```
alice@faui01[trunk/aufgabe1]$ svn commit
```


6 Aufbau des SP-Repositories



- Zur Abgabe wird ein Abgabezweig für jede Aufgabe in *branches* erzeugt
- Zur Vereinfachung der Abgabe bieten wir ein Skript

```
alice@faui01[aufgabe1]$ /proj/i4sp/bin/submit aufgabe1
```

  - dieses gibt die aktuelle HEAD-Version Ihres Repositories ab
  - offene Änderungen vor der Abgabe in den *trunk* einchecken
  - unterhalb von *branches* sollte nicht von Hand editiert/eingecheckt werden

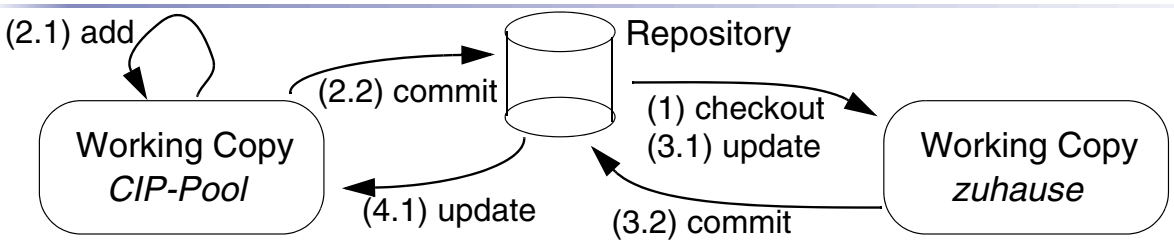
7 Abgabemodalitäten

- mehrmalige Abgabe ist möglich
  - durch erneuten Aufruf des *submit*-Skripts
- gewertet wird die letzte rechtzeitige Abgabe
- 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

```
alice@faui01[~] cd /proj/i4sp/alice/trunk
alice@faui01[trunk] mkdir aufgabe1
alice@faui01[trunk] cd aufgabe1
alice@faui01[aufgabe1] vim lilo.c
...
alice@faui01[aufgabe1] cd ..
alice@faui01[trunk] svn add aufgabe1
A   aufgabe1
A   aufgabe1/lilo.c
alice@faui01[trunk] svn commit
...
Committed revision 2.
alice@faui01[trunk] vim aufgabe1/lilo.c
...
alice@faui01[trunk] svn commit -m 'Bugfix in insertElement'
...
Committed revision 3.
alice@faui01[trunk] /proj/i4sp/bin/submit aufgabe1
...
# Aufgabe 1 ist jetzt abgegeben
```

9 Arbeiten zuhause



(1) Zusätzliche Arbeitskopien müssen zunächst erstellt werden (*checkout,* einmalig)

(2) Start der Arbeit an einer Aufgabe im CIP-Pool
- angelegte Dateien und Verzeichnisse unter Versionskontrolle stellen (*add*)
- Zwischenstand ins Repository einchecken (*commit*)

(3) Arbeit zuhause fortsetzen
- Arbeitskopie zunächst auf den aktuellen Stand bringen (*update*)
- Zwischenstand ins Repository einchecken (*commit*)

(4) Arbeit im CIP-Pool fortsetzen
- Arbeitskopie zunächst auf den aktuellen Stand bringen (*update*)
- ...

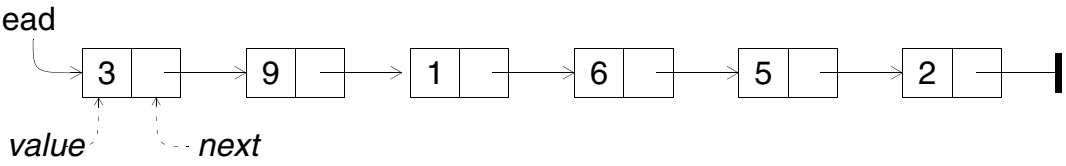
10 Hinweise zum Abgabesystem

- Ausführliche SVN-Dokumentation im Subversion-Buch
  http://svnbook.red-bean.com
- Wichtige Links
  - Subversion-Homepage: *http://www.subversion.org*
  - TortoiseSVN (Windows): *http://tortoisesvn.tigris.org*

U1-7 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