U1 1. Übung

- Allgemeines zum Übungsbetrieb
- Nachtrag zur Benutzerumgebung
- Ergänzungen zu C
  - Portable Programme
  - Gängige Compiler-Warnungen
  - Dynamische Speicherverwaltung
- Aufgabe 1

U1-1 Allgemeines zum Übungsbetrieb

1 Anlaufstellen

- Forum: `https://fsi.informatik.uni-erlangen.de/forum/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

- meist in Zweier-Teams (siehe Aufgabenstellung)
- Korrektur und Bewertung durch den jeweiligen Tafelübungsleiter
  - korrigierte Ausdrucke werden 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 Environment

- Das *Environment* eines Benutzers besteht aus einer Reihe von Text-Variablen, die an alle aufgerufenen Programme übergeben werden und von diesen abgefragt werden können

- Mit dem Kommando ***env(1)*** können die Werte der Environment-Variablen abgefragt werden:

```
% env
HOME=/home/jklein
LOGNAME=jklein
MANPATH=/local/man:/usr/man
PATH=/home/jklein/.bin:/local/bin:/usr/ucb:/bin:/usr/bin
SHELL=/bin/sh
TERM=vt100
USER=jklein
```

2 Environment

- Mit dem Kommando ***env(1)*** kann das Environment auch nur für ein Kommando gezielt verändert werden
- Auf Environment-Variablen kann – wie auf normale Shell-Variablen auch – durch **$*Variablenname*** in Kommandos zugegriffen werden
- Mit dem Kommando ***setenv(1)*** (C-Shell) bzw. ***set*** und ***export*** (Bourne Shell) können Environment-Variablen verändert und neu erzeugt werden:

```
% setenv PATH "$HOME/.bin:$PATH"

$ set PATH="$HOME/.bin:$PATH"; export PATH
```

2 Environment

- Überblick über einige wichtige Environment-Variablen

$USER	Benutzername (BSD)
$LOGNAME	Benutzername (SystemV)
$HOME	Homedirectory
$TERM	Terminaltyp (für bildschirmorientierte Programme, z. B. *emacs*)
$PATH	Liste von Directories, in denen nach Kommandos gesucht wird
$MANPATH	Liste von Directories, in denen nach Manual-Seiten gesucht wird (für Kommando ***man(1)***)
$SHELL	Dateiname des Kommandointerpreters (wird teilweise verwendet, wenn aus Programmen heraus eine Shell gestartet wird)
$DISPLAY	Angabe, auf welchem Rechner/Ausgabegerät das X-Windows-System seine Fenster darstellen soll

3 Die Environmentvariable $PATH

- Suchpfad für Anwendungsprogramme (durch ':' getrennt)

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin
$ ls
...
```

- Wird ein Kommando ohne explizite Pfadangabe gestartet, werden die Verzeichnisse im Suchpfad der Reihenfolge nach durchsucht
- Das erste gefundene Programm wird gestartet
- Unterschied zu DOS: '.' ist in der Regel nicht im Suchpfad enthalten
- Start von Programmen im akt. Verzeichnis mit expliziter Pfadangabe

```
$ ./slist
...
```

3 Intermezzo: Aktuelles Verzeichnis im Pfad

- '.' sollte aus Sicherheitsgründen nicht in $PATH aufgenommen werden
  - wenn es die Bequemlichkeit erfordert, '.' hinten in $PATH eintragen
- Angreifer legt ein Shell-Skript mit Namen `ls` in `/tmp` ab:

```
#!/bin/sh
echo muhaha
/bin/ls "$@"
```

- Opfer:

```
$ export PATH=".:$PATH"
$ cd /tmp
$ ls
muhaha
...
```

4 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-C89

(2) Verwenden einer standardisierten Betriebssystemschnittstelle
Hier: POSIX.1

1 ANSI-C

- Normierung des Sprachumfangs der Programmiersprache C
- Standard-Bibliotheksfunktionen
  (z. B. printf, malloc, …)
- gcc-Aufrufoptionen: `-ansi -pedantic`

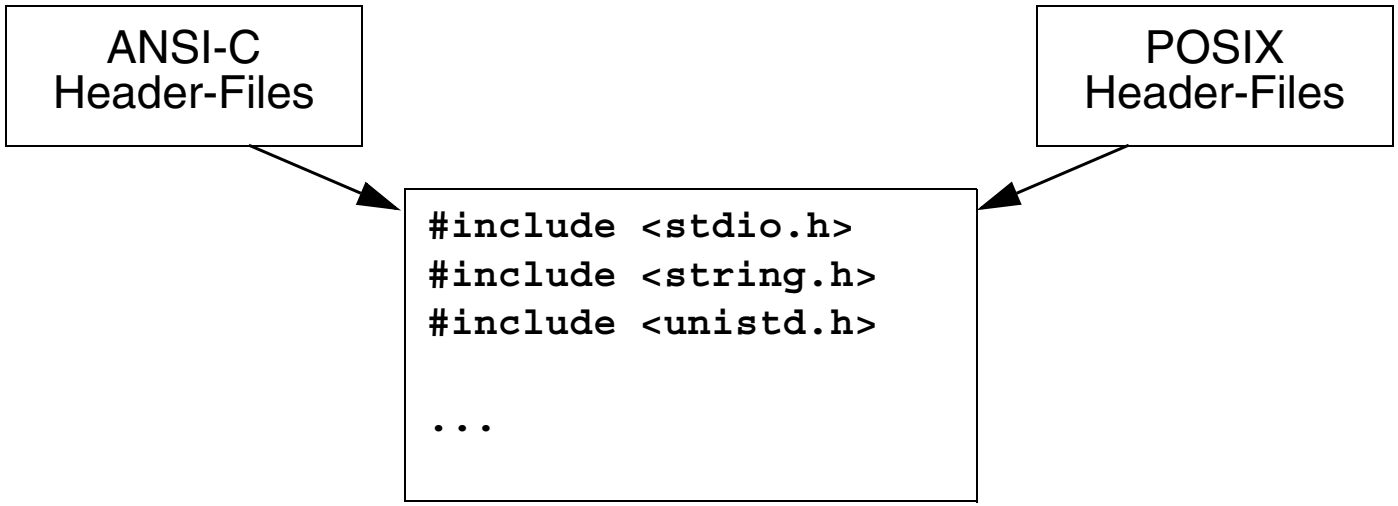
2 POSIX

- Standardisierung der Betriebssystemschnittstelle:
  Portable **O**perating **S**ystem **I**nterface (IEEE Standard 1003.1)
- POSIX.1 wird von verschiedenen Betriebssystemen implementiert:
  - SUN Solaris, SGI Irix, DIGITAL Unix, HP-UX, AIX
  - Linux
  - einige Versionen von Windows (*Windows Services for UNIX*)
  - Mac OS (Darwin)
- ...hier POSIX.1
  - gcc-Aufrufoption (empfohlen): `-D_POSIX_SOURCE`
  - alternativ **`#define`** im Programmtext (in jedem Modul, vor den Includes)

```
#define _POSIX_SOURCE
```

3 Header-Files: ANSI und POSIX

- In den Standards ANSI-C und POSIX.1 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 POSIX-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-C89
- Betriebssystemschnittstelle konform zu POSIX.1
- warnungs- und fehlerfrei mit folgendem Aufruf übersetzen (Bsp. `slist`):

```
gcc -ansi -pedantic -D_POSIX_SOURCE -Wall -Werror -o slist slist.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

- *C++ style comments are not allowed in ISO C90*
  - //-Kommentare sind im älteren C90-Standard nicht erlaubt
  - /* Nur solche Kommentare verwenden */
- *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
- *ISO C90 forbids mixed declarations and code*
  - in C90 dürfen Variablen nur am Anfang eines { . . }-Blocks deklariert werden
- *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 (2)

- Verlängern von Feldern, die durch malloc bzw. realloc erzeugt wurden:

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

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

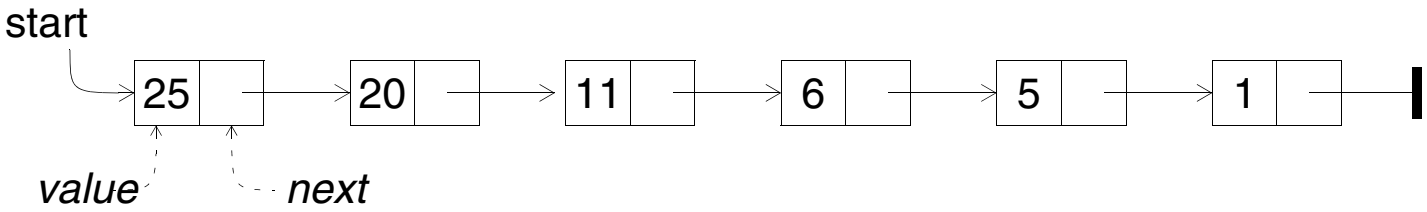
- Freigeben von Speicher

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

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

U1-6 1. Aufgabe

1 Sortierte verkettete Liste



- Strukturdefinition:

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

2 Funktionen

- **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 ein negativer Wert.
- **int removeElement(int value)**: Entfernt den Wert *value* aus der Liste und gibt diesen zurück. Ist der Wert in der Liste nicht vorhanden, wird ein negativer Wert zurückgeliefert.
- **int getMinVal(), int getMaxVal()**: Liefern Minimum bzw. Maximum der in der Liste gespeicherten Werte, oder ein negatives Ergebnis bei leerer Liste.
- **void printList()**: Gibt die in der Liste enthaltenen Werte auf die Standardausgabe in absteigender Reihenfolge aus, oder den String *(empty)* wenn die Liste keine Werte enthält.