

# U1 1. Übung

---

## U1-1 Überblick

---

- Ergänzungen zu C
  - ◆ Dynamische Speicherverwaltung
  - ◆ Portable Programme
- Aufgabe 1
- UNIX-Benutzerumgebung und Shell
- UNIX-Kommandos

# U1-2 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(person == 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);
```

# U1-2 Dynamische Speicherverwaltung (2)

- Verlängern von Felder, 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!

## U1-3 Portable Programme

- 1. Verwenden der standardisierten Programmiersprache ANSI-C

- ◆ gcc-Aufrufoptionen

```
-ansi -pedantic
```

- 2. Verwenden einer standardisierten Betriebssystemschnittstelle, z.B. POSIX

- ◆ gcc-Aufrufoption

```
-D_POSIX_SOURCE
```

- ◆ oder `#define` im Programmtext

```
#define _POSIX_SOURCE
```

- Programm sollte sich mit folgenden gcc-Aufruf compilieren lassen

```
gcc -ansi -pedantic -D_POSIX_SOURCE -Wall -Werror
```

# 1 POSIX

---

- Standardisierung der Betriebssystemschnittstelle:  
**P**ortable **O**perating **S**ystem Interface (IEEE Standard 1003.1)
- POSIX.1 wird von verschiedenen Betriebssystemen implementiert:
  - ◆ SUN Solaris, SGI Irix, DIGITAL Unix, HP-UX, AIX
  - ◆ Linux
  - ◆ Windows (POSIX Subsystem)
  - ◆ ...

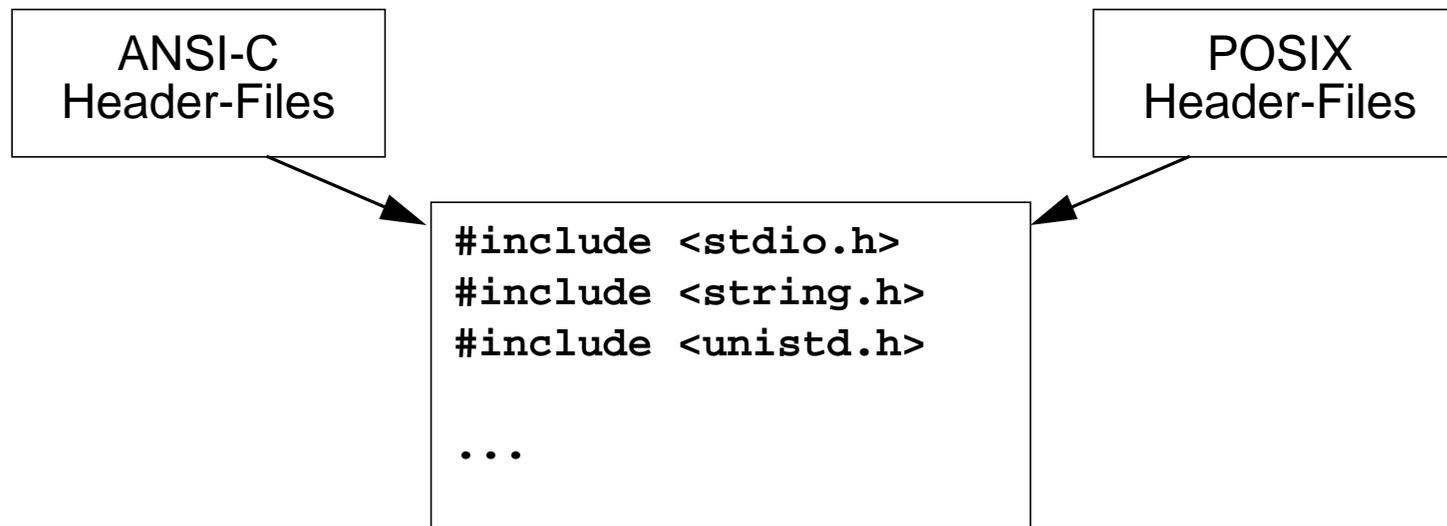
# 2 ANSI-C

---

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

### 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 ANSI-C Header-Files

---

- **assert.h**: assert()-Makro
- **ctype.h**: Makros und Funktionen für Characters (z.B. tolower(), isalpha())
- **errno.h**: Fehlerauswertung (z.B. errno-Variable)
- **float.h**: Makros für Fließkommazahlen
- **limits.h**: Enthält Definitionen für Systemschranken
- **locale.h**: Funktion setlocale()
- **math.h**: Mathematische Funktionen für double
- **setjmp.h**: Funktionen setjmp(), longjmp()
- **signal.h**: Signalbehandlung
- **stdarg.h**: Funktionen und Makros für variable Argumentlisten
- **stddef.h**: Def. von ptrdiff\_t, NULL, size\_t, wchar\_t, offsetpf, errno
- **stdio.h**: I/O Funktionen (z.B. printf(), scanf(), fgets())
- **stdlib.h**: Hilfsfunktionen (z.B. malloc(), getenv(), rand())
- **string.h**: Stringmanipulation (z.B. strcpy())
- **time.h**: Zeitmanipulation (z.B. time(), ctime(), strftime())

## 5 POSIX Header-Files

- **dirent.h**: opendir(), readdir(), rewinddir(), closedir()
- **fcntl.h**: open(), creat(), fcntl()
- **grp.h**: getgrgid(), getgrnam()
- **pwd.h**: getpwuid(), getpwnam()
- **setjmp.h**: sigsetjmp(), siglongjmp()
- **signal.h**: kill(), sigemptyset(), sigfillset(), sigaddset(), sigdelset(), sigismember(), sigaction, sigprocmask(), sigpending(), sigsuspend()
- **stdio.h**: ctermid(), fileno(), fdopen()
- **sys/stat.h**: umask(), mkdir(), mkfifo(), stat(), fstat(), chmod()
- **sys/times.h**: times()
- **sys/types.h**: enthält betriebssystemabhängige Typdefinitionen
- **sys/utsname.h**: uname()
- **sys/wait.h**: wait(), waitpid()
- **termios.h**: cfgetospeed(), cfsetospeed(), cfgetispeed(), cfsetispeed(), tcgetattr(), tcsetattr(), tcsendbreak(), tcdrain(), tcflush(), tcflow()
- **time.h**: time(), tzset()
- **utime.h**: utime()
- **unistd.h**: alle POSIX-Funktionen, die nicht in den obigen Header-Files definiert sind (z.B. fork(), read())

## 6 POSIX Datentypen

### ■ Typ-Deklarationen über typedef-Anweisung — Beispiel

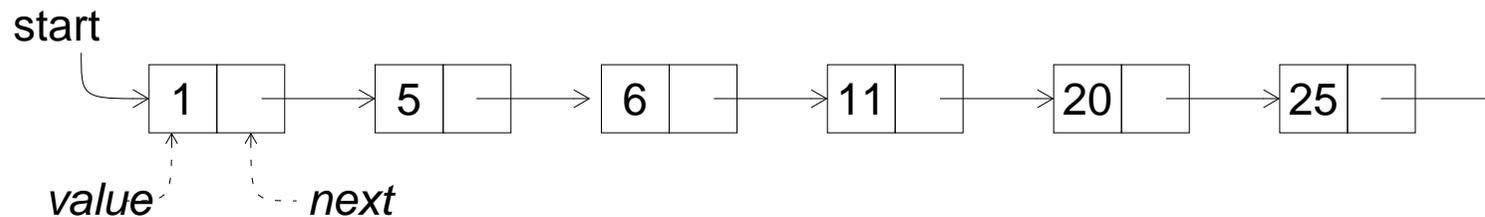
```
typedef unsigned long dev_t;  
dev_t device;
```

### ■ 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

# U1-4 1. Aufgabe

## 1 Warteschlange als verkettete Liste



### ■ Strukturdefinition:

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

### ■ Funktionen:

- ◆ `void append_element(int)`: Anfügen eines Elements ans Listenende
- ◆ `int remove_element()`: Entnehmen eines Elements vom Listenanfang

## U1-5 Benutzerumgebung

---

- die voreingestellte Benutzerumgebung umfasst folgende Punkte:
  - Benutzername
  - Identifikation (**User-Id und Group-Ids**)
  - Home-Directory
  - Shell

## U1-6 Sonderzeichen

---

- einige Zeichen haben unter UNIX besondere Bedeutung
- Funktionen:
  - Korrektur von Tippfehlern
  - Steuerung der Bildschirm-Ausgabe
  - Einwirkung auf den Ablauf von Programmen

## U1-6 Sonderzeichen (2)

- die Zuordnung der Zeichen zu den Sonderfunktionen kann durch ein UNIX-Kommando (***stty(1)***) verändert werden
- die Vorbelegung der Sonderzeichen ist in den verschiedenen UNIX-Systemen leider nicht einheitlich
- Übersicht:
 

<BACKSPACE>	letztes Zeichen löschen (häufig auch <DELETE>)
<DELETE>	alle Zeichen der Zeile löschen (häufig auch <CTRL>U oder <CTRL> X)
<CTRL>C	Interrupt - Programm wird abgebrochen
<CTRL>\	Quit - Programm wird abgebrochen + core-dump
<CTRL>Z	Stop - Programm wird gestoppt (nicht in sh)
<CTRL>D	End-of-File
<CTRL>S	Ausgabe am Bildschirm wird angehalten
<CTRL>Q	Ausgabe am Bildschirm läuft weiter

# U1-7 UNIX-Kommandointerpreter: Shell

---

auf den meisten Rechnern stehen verschiedene Shells zur Verfügung:

- sh**      **Bourne-Shell** - erster UNIX-Kommandointerpreter  
(wird vor allem für Kommandoprozeduren verwendet)
- ksh**      **Korn-Shell** - ähnlich wie Bourne-Shell, aber mit eingebautem Zeileneditor  
(vi- oder emacs-Modus)
- csh**      **C-Shell** (stammt aus der Berkeley-UNIX-Linie) - vor allem für interaktive  
Benutzung geeignet
- tcsh**     **erweiterte C-Shell** - enthält zusätzliche Editier-Funktionen, ähnlich wie  
Korn-Shell
- bash**     Shell der GNU-Distribution (*Bourne-Again Shell*)

# 1 Aufbau eines UNIX-Kommandos

---

UNIX-Kommandos bestehen aus:

## ■ Kommandonamen

(der Name einer Datei in der ein ausführbares Programm oder eine Kommandoprozedur für die Shell abgelegt ist)

## ■ einer Reihe von **Optionen** und **Argumenten**

- Kommandoname, Optionen und Argumente werden durch Leerzeichen oder Tabulatoren voneinander getrennt
- Optionen sind meist einzelne Zeichen hinter einem Minus(-)-Zeichen
- Argumente sind häufig Namen von Dateien, die von dem Kommando bearbeitet werden

Nach dem Kommando wird automatisch in allen Directories gesucht, die in der *Environment-Variablen* **\$PATH** aufgelistet sind.

!!! Sicherheitsprobleme wenn das aktuelle Directory im Pfad ist (Trojanische Pferde)

## 2 Vordergrund- / Hintergrundprozess

---

- die Shell meldet mit einem Promptsymbol (z. B. `faui09%`), dass sie ein Kommando entgegennehmen kann
- die Beendigung des Kommandos wird abgewartet, bevor ein neues Promptsymbol ausgegeben wird - **Vordergrundprozess**
- wird am Ende eines Kommandos ein **&**-Zeichen angehängt, erscheint sofort ein neues Promptsymbol - das Kommando wird im Hintergrund bearbeitet - **Hintergrundprozess**

## 2 Vordergrund- / Hintergrundprozess (2)

### ■ Jobcontrol:

- durch <CTRL>Z kann die Ausführung eines Kommandos (*Job*) angehalten werden - es erscheint ein neues Promptsymbol
- funktioniert nicht in der *Bourne-Shell*

### ■ die Shell (*cs**h*, *tc**sh*, *k**sh*, *b**ash*) stellt einige Kommandos zur Kontrolle von Hintergrundjobs und gestoppten Jobs zur Verfügung:

<b>jobs</b>	Liste aller existierenden Jobs
<b>bg %n</b>	setze Job <b>n</b> im Hintergrund fort
<b>fg %n</b>	hole Job <b>n</b> in den Vordergrund
<b>stop %n</b>	stoppe Hintergrundjob <b>n</b>
<b>kill %n</b>	beende Job <b>n</b>

### 3 Ein- und Ausgabe eines Kommandos

---

- jedes Programm wird beim Aufruf von der Shell mit 3 E/A-Kanälen versehen:
  - stdin** Standard-Eingabe (Vorbelegung = Tastatur)
  - stdout** Standard-Ausgabe (Vorbelegung = Terminal)
  - stderr** Fehler-Ausgabe (Vorbelegung = Terminal)
  
- diese E/A-Kanäle können auf Dateien umgeleitet werden oder auch mit denen anderer Kommandos verknüpft werden (**Pipes**)

## 4 Umlenkung der E/A-Kanäle auf Dateien

- die Standard-E/A-Kanäle eines Programms können von der Shell aus umgeleitet werden  
(z. B. auf reguläre Dateien oder auf andere Terminals)
- die Umleitung eines E/A-Kanals erfolgt in einem Kommando (am Ende) durch die Zeichen `<` und `>`, gefolgt von einem Dateinamen
- durch `>` wird die Datei ab Dateianfang überschrieben, wird statt dessen `>>` verwendet, wird die Kommandoausgabe an die Datei angehängt

- Syntax-Übersicht

<code>&lt;datei1</code>	legt den Standard-Eingabekanal auf <b>datei1</b> , d. h. das Kommando liest von dort
<code>&gt;datei2</code>	legt den Standard-Ausgabekanal auf <b>datei2</b>
<code>&gt;&amp;datei3</code>	( <i>cs</i> h, <i>tc</i> sh) legt Standard- und Fehler-Ausgabe auf <b>datei3</b>
<code>2&gt;datei4</code>	( <i>sh</i> , <i>ksh</i> , <i>bash</i> ) legt den Fehler-Ausgabekanal auf <b>datei4</b>
<code>2&gt;&amp;1</code>	( <i>sh</i> , <i>ksh</i> , <i>bash</i> ) verknüpft Fehler- mit Standard-Ausgabekanal (Unterschied zu " <code>&gt;datei 2&gt;datei</code> " !!!)

## 5 Pipes

- durch eine **Pipe** kann der Standard-Ausgabekanal eines Programms mit dem Eingabekanal eines anderen verknüpft werden
- die Kommandos für beide Programme werden hintereinander angegeben und durch | getrennt

- Beispiel:

```
ls -al | wc
```

- ▶ das Kommando **wc** (Wörter zählen), liest die Ausgabe des Kommandos **ls** und gibt die Anzahl der Wörter (Zeichen und Zeilen) aus
- *Csh* und *tcsh* erlauben die Verknüpfung von Standard-Ausgabe und Fehler-Ausgabe in einer Pipe:
  - ▶ Syntax: **|&** statt **|**

## 6 Kommandoausgabe als Argumente

---

- die Standard-Ausgabe eines Kommandos kann einem anderen Kommando als Argument gegeben werden, wenn der Kommandoaufruf durch `` `` geklammert wird

- Beispiel:

```
rm `grep -l XXX *`
```

- ◆ das Kommando `grep -l XXX` liefert die Namen aller Dateien, die die Zeichenkette **XXX** enthalten auf seinem Standard-Ausgabekanal
  - ↳ es werden alle Dateien gelöscht, die die Zeichenkette **XXX** enthalten

## 7 Quoting

---

Wenn eines der Zeichen mit Sonderbedeutung (wie `<`, `>`, `&`) als Argument an das aufzurufende Programm übergeben werden muß, gibt es folgende Möglichkeiten dem Zeichen seine Sonderbedeutung zu nehmen:

- 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

## 8 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
EXINIT=se aw ai sm
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
TTY=/dev/pts/1
USER=jklein
HOST=fau43d
```

## 8 Environment (2)

- 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** (Shell) können Environment-Variablen verändert und neu erzeugt werden:

```
% setenv PATH "$HOME/.bin.sun4:$PATH"
```

```
$ set PATH="$HOME/.bin.sun4:$PATH"; export PATH
```

## 8 Environment (2)

- Überblick über einige wichtige Environment-Variablen

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

# U1-8 UNIX-Kommandos

---

- man-Pages
- Dateisystem
- Benutzer
- Prozesse
- diverse Werkzeuge

# 1 man-Pages

---

- Aufgeteilt nach verschiedenen *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`

## 2 Dateisystem

---

<b>ls</b>	Directory auflisten wichtige Optionen: -l langes Ausgabeformat -a auch mit . beginnende Dateien werden aufgeführt
<b>chmod</b>	Zugriffsrechte einer Datei verändern
<b>cp</b>	Datei(en) kopieren
<b>mv</b>	Datei(en) verlagern (oder umbenennen)
<b>ln</b>	Datei linken (weiteren Verweis auf gleiche Datei erzeug.)
<b>ln -s</b>	Symbolic link erzeugen
<b>rm</b>	Datei(en) löschen
<b>mkdir</b>	Directory erzeugen
<b>rmdir</b>	Directory löschen (muß leer sein!!!)

### 3 Benutzer

---

<b>id, groups</b>	eigene Benutzer-Id und Gruppenzugehörigkeit ausgeben
<b>who</b>	am Rechner angemeldete Benutzer
<b>finger</b>	ausführlichere Information über angemeldete Benutzer
<b>finger user@faiu02</b>	Info über Benutzer am CIP-Pool

## 4 Prozesse

---

<b>ps</b>		Prozessliste ausgeben
	<b>-u x</b>	Prozesse des Benutzers x
	<b>-ef</b>	alle Prozesse (-e), ausführliches Ausgabeformat (-f)
<b>top</b>		Prozessliste, sortiert nach aktueller Aktivität
<b>kill &lt;pid&gt;</b>		Prozess "abschießen" (Prozess kann aber bei Bedarf noch aufräumen oder den Befehl sogar ignorieren)
<b>kill -9 &lt;pid&gt;</b>		Prozess "gnadenlos abschießen" (Prozess hat keine Chance)

## 5 diverse Werkzeuge

---

<b>cat</b>	Datei(en) hintereinander ausgeben
<b>more, less</b>	Dateien bildschirmweise ausgeben
<b>head</b>	Anfang einer Datei ausgeben (Vorbel. 10 Zeilen)
<b>tail</b>	Ende einer Datei ausgeben (Vorbel. 10 Zeilen)
<b>pr, lp, lpr</b>	Datei ausdrucken
<b>wc</b>	Zeilen, Wörter und Zeichen zählen
<b>grep, fgrep, egrep</b>	nach bestimmten Mustern bzw. Zeichenketten suchen
<b>find</b>	Dateibaum traversieren
<b>sed</b>	Stream-Editor
<b>tr</b>	Zeichen abbilden
<b>awk</b>	pattern-scanner
<b>cut</b>	einzelne Felder aus Zeilen ausschneiden
<b>sort</b>	sortieren