Übungen zu Systemprogrammierung 1 (SP1)

Ü2 – Sortieren und Tooling

Jens Schedel, Christoph Erhardt, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 2015 – 24. bis 30. April 2015

https://www4.cs.fau.de/Lehre/SS15/V_SP1

Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

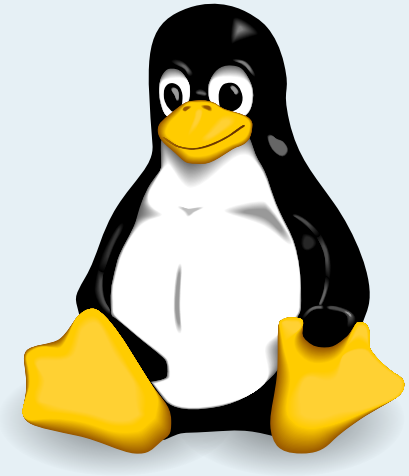
Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Veranstaltungshinweis

Linux-Install-Party der FSI

- am Mittwoch, den 06.05.2015, um 14:00
- im 02.152-113 (Blaues Hochhaus, 2. Stock)
- weitere Informationen unter
  `https://fsi.informatik.uni-erlangen.de/linuxinstall`



Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Anforderungen an abgegeben Lösungen

- C-Sprachumfang konform zu ANSI C99
- Betriebssystemschnittstelle konform zu SUSv4
- **warnungs-** und **fehlerfrei** im CIP-Pool mit folgenden `gcc`-Optionen übersetzen

```
-std=c99 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror
```

  - `-std=c99 -pedantic` erlauben nur ANSI-C99-konformen C-Quellcode
  - `-D_XOPEN_SOURCE=700` erlaubt nur SUSv4-konforme Betriebssystemaufrufe

- einzelne Aufgaben können hiervon abweichen, dies wird in der Aufgabenstellung entsprechend vermerkt

Versionierungsschema

- Subversion nummeriert fortlaufend ab Revision 0 (1, 2, 3, ...)
- spezielle Revisionsschlüsselwörter
  - `HEAD`: aktuelle Version des Repositories (neueste Version)
  - `BASE`: Revision eines Eintrags (Datei oder Verzeichnis) der Arbeitskopie
  - `COMMITTED`: Letzte Änderungsrevision eines Eintrags – meist älter als `BASE`
  - `PREV`: `COMMITTED - 1`
- Revision zu einem bestimmten Zeitpunkt
  - `“{2015-05-11 17:30}“`

Basisoperationen II

- `diff`: Änderungen der Arbeitskopie anzeigen

```
> svn status
M hallo
> svn diff
Index: hallo
===================================================================
--- hallo (revision 23)
+++ hallo (working copy)
@@ -0,0 +1 @@
+welt
```

- `revert`: Noch nicht committete Änderungen an der Arbeitskopie zurücksetzen

```
> svn revert hallo
Reverted 'hallo'
> svn status
>
```

Basisoperationen II

- `list/ls`: Dateien/Verzeichnisse im Repository anzeigen

```
> svn ls
branches/
trunk/
```

- `log`: Historie anzeigen

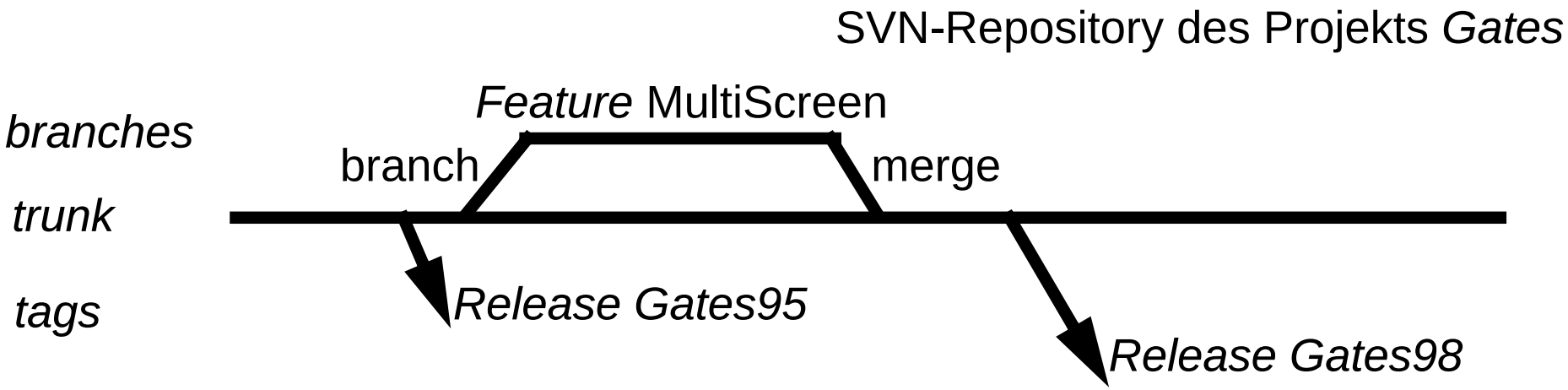
```
> svn log
------------------------------------------------------------------------
r1 | www-data | 2013-04-19 15:03:14 +0200 (Fri, 19 Apr 2013) | 1 line

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

- `move/mv`: Datei umbenennen oder verschieben
- `copy/cp`: Datei/Teilbaum kopieren

```
> svn cp aufgabe2 contest
> # aufgabe2 wurde in contest kopiert
```

Konventionelles Repository-Layout



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

Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Abgabesystem: Team-Arbeit

- Gemeinsame Bearbeitung im Repository eines Teammitglieds
  - Repository-Eigentümer: *alice*
  - Partner (nutzt Repository von *alice*): *bob*
- Abgabe erfolgt ebenfalls im Repository des Eigentümers
  - es ist nur eine Abgabe erforderlich

Ablauf für den Repository-Eigentümer

- Partner wird für jede Team-Aufgabe separat festgelegt

```
> /proj/i4sp1/bin/set-partner aufgabe2 bob
```

- Hintergrund
  - Erzeugung und Commit einer Textdatei `partner` in `trunk/aufgabe2`
  - diese Datei enthält den Login-Namen (*bob*) des Partners für diese Aufgabe
  - Partner erhält Zugriff auf die relevanten Teile des Repositorys
    - `trunk/aufgabe2`
    - `branches/aufgabe2`

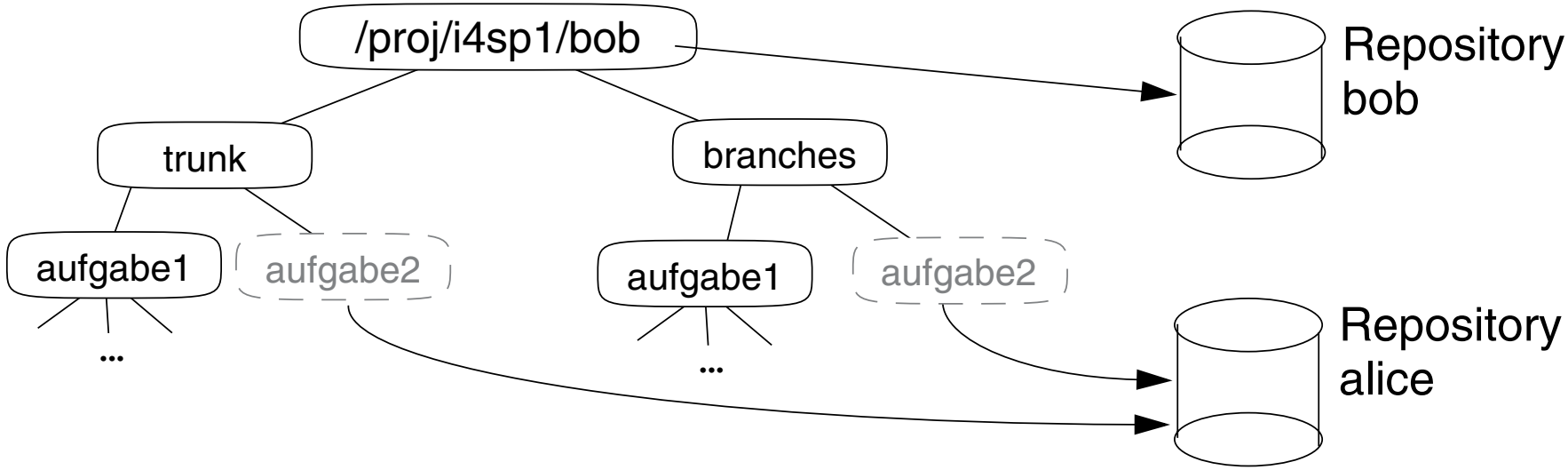
- Abgabe funktioniert wie gewohnt

```
> /proj/i4sp1/bin/submit aufgabe2
```

Ablauf für den Partner

- Partner setzt in seinem Repository einen Verweis auf Hauptrepository

```
> /proj/i4sp1/bin/import-from-partner aufgabe2 alice
```



- Nach Ausführung des Skriptes `svn update` ausführen
- Achtung: Abgabe im eigenen Repository überlagert Partnerabgabe
  - zum Umstieg auf Teamarbeit eigene Abgabe löschen (Übungsleiter hilft)

Ablauf für den Partner

- Arbeit in der eigenen Arbeitskopie fast normal möglich
  - Der Befehl `svn commit` übermittelt nur Änderungen an das Repository das für den aktuellen Pfad zuständig ist

```
> svn status
X       aufgabe2
M       aufgabe2/wsort.c
> svn commit
# Es erfolgt kein Commit
> svn status
X       aufgabe2
M       aufgabe2/wsort.c
> cd aufgabe2
> svn commit -m 'Erste Arbeiten an Aufgabe 2 abgeschlossen'
Committed revision 5.
```

- Abgabe funktioniert wie gewohnt

```
> /proj/i4sp1/bin/submit aufgabe2
```

Dynamische Speicherverwaltung

- Rückblick: Aufgabe 10.1 (Menschenkette) in AuD SS 2012

```
WaitingHuman somebody = new WaitingHuman("Mrs. Somebody");
```

  - In Java: Neues Listenelement wird mit Hilfe von `new` instanziiert
    - Reservieren eines Speicherbereiches für das Objekt
    - Initialisieren des Objektes durch Ausführen des Konstruktors

- In C: Anlegen eines Listenelementes mittels `malloc(3)`

```
struct listelement *newElement;
newElement = malloc( sizeof(struct listelement) );
if( newElement == NULL ) {
    // Fehlerbehandlung
}
```

    - Zurückgegebener Speicher hat undefinierten/zufälligen Wert
    - Inititialisierung muss per Hand erfolgen

Dynamische Speicherverwaltung

- Explizite Initialisierung mit definiertem Wert: `memset(3)`

```
memset(newElement, 0, sizeof(struct listelement));
```

- Mit 0 vorinitialisierter Speicher kann mit `calloc(3)` angefordert werden

```
struct listelement *newElement;
newElement = calloc( 1, sizeof(struct listelement) );
if ( newElement == NULL ) { /* Fehler */ }
```

- Im Gegensatz zu Java gibt es in C keinen Garbage-Collection-Mechanismus
  - Speicherbereich muss von Hand mittels `free(3)` freigegeben werden
  - Nur Speicher, der mit einer der Funktionen `malloc(3)`, `calloc(3)` oder `realloc(3)` angefordert wurde, darf mit `free(3)` freigegeben werden!
  - Zugriff auf freigegebenen Speicherbereich ist undefiniert

Dynamische Speicherverwaltung – Teil 2

- Verändern der Größe von Feldern, die durch `malloc(3)` bzw. `calloc(3)` erzeugt wurden:

```
int* numbers = malloc( n*sizeof(int) );
if ( numbers == NULL ) {
  // Fehlerbehandlung
}
... // Speicherbedarf gestiegen
int* neu = realloc( numbers, (n+10) * sizeof(int));
if(neu == NULL) {
  free(numbers);
  // Fehlerbehandlung
}
numbers = neu;
```

- Neuer Speicherbereich enthält die Daten des ursprünglichen Speicherbereichs (wird automatisch kopiert; aufwändig)
- Sollte `realloc(3)` fehlschlagen, wird der ursprüngliche Speicherbereich nicht freigegeben
  - Explizite Freigabe mit `free(3)` notwendig.

Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Fehlerbehandlung

- Fehler können aus unterschiedlichsten Gründen im Programm auftreten
  - Systemressourcen erschöpft: `malloc(3)` schlägt fehl
  - Fehlerhafte Benutzereingaben: `fopen(3)` schlägt fehl
  - Transiente Fehler: z. B. nicht erreichbarer Server
  - ...
- Gute Software **erkennt Fehler**, führt eine **angebrachte Behandlung** durch und gibt eine **aussagekräftige Fehlermeldung** aus
- Kann das Programm trotz des Fehlers sinnvoll weiterlaufen?
  - Beispiel 1: Benutzer gibt ungültige URL in den Browser ein
    - Fehlerbehandlung: Fehlermeldung anzeigen, Programm läuft weiter
  - Beispiel 2: Kopierprogramm: Öffnen der Quelldatei schlägt fehl
    - Fehlerbehandlung: Fehlermeldung anzeigen, Kopieren nicht möglich, Programm beenden

Fehlerbehandlung: Beendigung des Programms

- Tritt ein Fehler auf, der ein sinnvolles Weiterarbeiten verhindert, muss das Programm beendet werden (`exit(2)`) und einen Programmabbruch anzeigen.
- Signalisierung des Fehlers an Aufrufer des Programms über den Exitstatus
  - Exitstatus 0 zeigt erfolgreiche Programmausführung an
  - Werte ungleich 0 zeigen einen Fehler bei der Ausführung an
    - Die Bedeutung des entsprechenden Wertes ist nicht standardisiert
    - Manchmal enthält die Man-Page Informationen über die Bedeutung des Exitstatus
- libc bietet vordefinierte Makros für den Exitstatus an:
  - `EXIT_SUCCESS`
  - `EXIT_FAILURE`
- Exitstatus des letzten Befehls ist in der Shell-Variable `$?` gespeichert

Erkennung und Ausgabe von Fehlern

- Fehler treten häufig in Funktionen der C-Bibliothek auf
  - erkennbar i. d. R. am Rückgabewert (Man-Page, **RETURN VALUES**)
- Die Fehlerursache wird über die globale Variable `errno` übermittelt
  - Der Wert `errno = 0` ist reserviert, alles andere ist ein Fehlercode
  - Bibliotheksfunktionen setzen `errno` im Fehlerfall (sonst nicht zwingend)
  - Bekanntmachung im Programm durch Einbinden von `errno.h`
- Fehlercodes als lesbare Strings ausgegeben mit `perror(3)`

```
char *mem = malloc(...);      // malloc gibt im Fehlerfall
if(NULL == mem) {             // NULL zurück
    perror("malloc");         // Ausgabe der Fehlerursache
    exit(EXIT_FAILURE);       // Programm mit Fehlercode beenden
}
```

  - `perror(3)` nur verwenden, wenn die `errno` gesetzt wurde
  - sonst mit Hilfe von `fprintf(3)` eigene Fehlermeldung ausgeben

Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Generisches Sortieren mit qsort

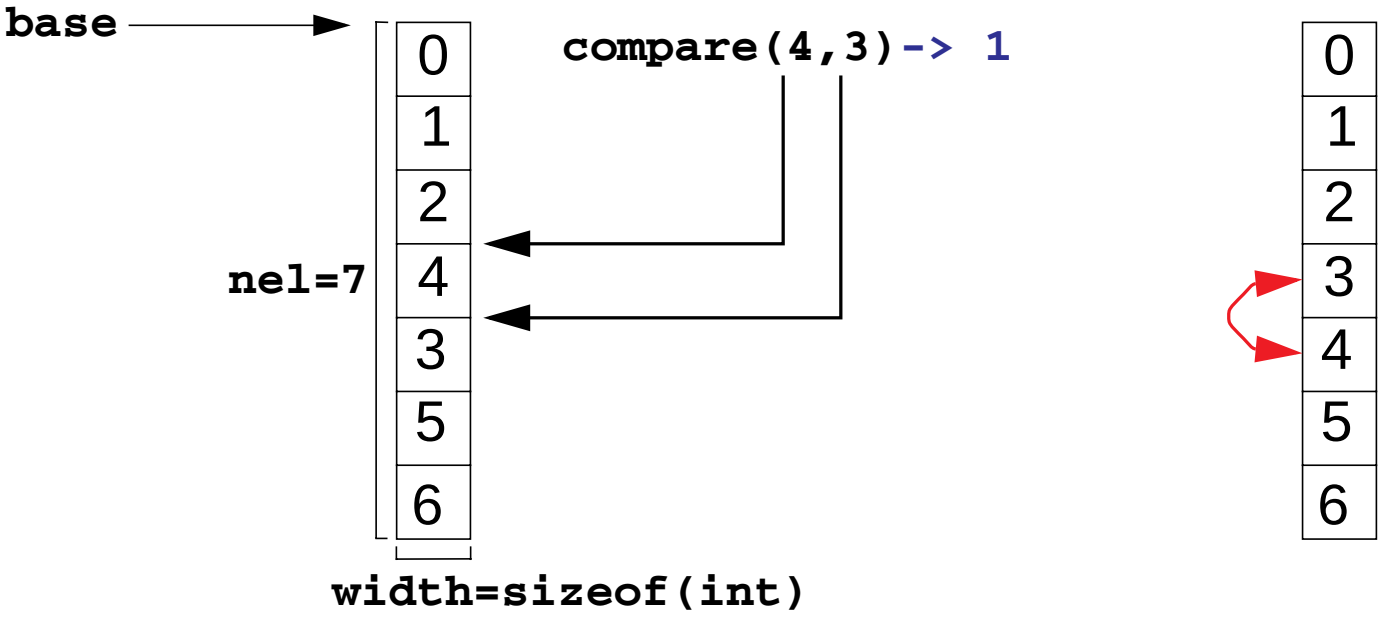
- Vergleich nahezu beliebiger Daten
  - alle Daten müssen die gleiche Größe haben
- `qsort` weiß nicht, was es sortiert (wie der Vergleich zu bewerkstelligen ist)
  - Aufrufer stellt Routine zum Vergleich zweier Elemente zur Verfügung
  - Fachbegriff für dieses Programmierschema: *Rückruf* (*Callback*)
- Prototyp aus `stdlib.h`:

```
void qsort(void *base,
           size_t nel,
           size_t width,
           int (*compare) (const void *, const void *));
```

  - `base`: Zeiger auf das erste Element des zu sortierenden Feldes
  - `nel`: Anzahl der Elemente im zu sortierenden Feld
  - `width`: Größe eines Elements
  - `compare`: Vergleichsfunktion

Arbeitsweise von qsort

- `qsort` vergleicht je zwei Elemente mit Hilfe der Vergleichsfunktion



- Die Funktion vergleicht die beiden Elemente und liefert:

`< 0` falls Element 1 kleiner gewertet wird als Element 2
`0` falls Element 1 und Element 2 gleich gewertet werden
`> 0` falls Element 1 größer gewertet wird als Element 2

Vergleichsfunktion

- Die Vergleichsfunktion erhält Zeiger auf Feldelemente
  - Beispiel: Vergleichsfunktion für int

```
int intCompare(const int *, const int *);
```

    - `const`-Zusicherung: Funktion ändert die verglichenen Werte nicht

- `qsort(3)` kennt den tatsächlichen Datentyp nicht
  - Prototyp ist generisch mit void-Zeigern parametrisiert

```
void qsort(...,int (*compare) (const void *, const void *));
```

  - Typumwandlung erforderlich

Vergleichsfunktion

- Typumwandlung abhängig vom Typ des zu sortierenden Datums
- explizit bei Zeiger-Datentypen
  - Hier: Sortieren eines Feldes von `int*`

```
static int wrap_intPointerCompare(const void *a, const void *b) {
  int * const *x = (int * const *)a;
  int * const *y = (int * const *)b;
  return intCompare(*x,*y);
}
```

- sonst implizit
  - Hier: Sortieren eines Feldes von `int`

```
static int wrap_intCompare(const void *a, const void *b) {
  return intCompare(a,b);
}
```

Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Zeilenweise Ein- und Ausgabe

Zeilenweises Lesen

```
char *fgets(char *s, int n, FILE *fp)
```

- liest Zeichen von Dateikanal `fp` in das Feld `s` bis entweder `n-1` Zeichen gelesen wurden oder `\n` gelesen oder EOF erreicht wurde
- `s` wird mit `\0` abgeschlossen (\n wird nicht entfernt)
- gibt bei EOF oder Fehler `NULL` zurück, sonst `s`
- für `fp` kann `stdin` eingesetzt werden, um von der Standardeingabe zu lesen

Zeilenweises Schreiben

```
int fputs(char *s, FILE *fp)
```

- schreibt die Zeichen im Feld `s` auf Dateikanal `fp`
- für `fp` kann auch `stdout` oder `stderr` eingesetzt werden
- als Ergebnis wird die Anzahl der geschriebenen Zeichen geliefert

Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Aufgabe 2: wsort

- Lernziele
  - Einlesen von der Standardeingabe (`stdin`)
  - Umgang mit dynamischer Speicherverwaltung (`realloc(3)`)
  - Verwendung von Debug-Werkzeugen (siehe Folien kommende Woche)
- Ausprobieren eures Programmes
  - Beispiel-Eingabedateien in `/proj/i4sp1/pub/aufgabe2`
  - Vergleichen der Ausgabe mit vorgegebenem Binary
    - Hier am Beispiel der `wlist0`

```
$ ./wsort < /proj/i4sp1/pub/aufgabe2/wlist0 > wlist0.mine
$ /proj/i4sp1/pub/aufgabe2/wsort < \
                  /proj/i4sp1/pub/aufgabe2/wlist0 > wlist0.spteam
$ diff wlist0.mine wlist0.spteam
```

    - Auch graphische Diff-Werkzeuge möglich: `kompare`, `meld`, etc.

Agenda

2.1 Linux-Install-Party der FSI
2.2 Anforderungen an Abgaben
2.3 Subversion – Teil 2
2.4 Abgabesystem: Team-Arbeit
2.5 Fehlerbehandlung
2.6 Generisches Sortieren
2.7 Ein- und Ausgabe
2.8 Aufgabe 2: wsort
2.9 Gelerntes anwenden

Aktive Mitarbeit!

„Aufgabenstellung“

- Personalisiertes Hello World schreiben
- `isort` Programm, welches als Argumente übergebene Zahlen sortiert