Übungen Betriebssysteme (BS)

U0 – Erste Schritte in C

https://moodle.tu-dortmund.de/course/view.php?id=34604

Peter Ulbrich

peter.ulbrich@tu-dortmund.de

https://sys.cs.tu-dortmund.de/EN/People/ulbrich/

technische universität
dortmund

Theoriefrage 1

Mit welchen Parametern zeigt `ls` den gesamten Inhalt eines Verzeichnisses im (langen) Listenformat an?

Theoriefrage 1

Mit welchen Parametern zeigt `ls` den gesamten Inhalt eines Verzeichnisses im (langen) Listenformat an?

```
studi@bsvm:~$ ls -lA
insgesamt 32
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 Bilder
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 .config
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 Dokumente
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 Downloads
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 .local
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 Musik
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 Öffentlich
drwxr-xr-x 2 studi studi 4096 Mai 22  2020 Schreibtisch
drwxr-xr-x 2 studi studi 4096 Apr  7  2020 Videos
...
```

Theoriefrage 2

Erklärt in eigenen Worten, wofür man das UNIX-Kommando `man` verwendet. Erklärt den Unterschied zwischen den Befehlen `man 1 printf` und `man 3 printf` und was hier beschrieben wird.

Theoriefrage 2

Erklärt in eigenen Worten, wofür man das UNIX-Kommando `man` verwendet. Erklärt den Unterschied zwischen den Befehlen `man 1 printf` und `man 3 printf` und was hier beschrieben wird.

`man` gibt die Handbuchseite/Anleitung (Manual) zu dem Befehl oder der Funktion oder Datei an.

Die Nummer hinter **`man`** bestimmt, in welchem Themenbereich/welcher Section gesucht werden soll.

`man 1 printf` beschreibt das Shell-Kommando `printf`.

`man 3 printf` beschreibt die C-Funktion printf aus `stdio.h`

Theoriefrage 3

Mit welchem UNIX-Kommando kann man Dateien und Ordner umbenennen? Für welchen Zweck kann man dieses Kommando noch verwenden?

Theoriefrage 3

Mit welchem UNIX-Kommando kann man Dateien und Ordner umbenennen? Für welchen Zweck kann man dieses Kommando noch verwenden?

`mv alter-name neuer-name`	Umbenennen
`mv foo ~/Downloads/`	Verschieben

```
studi@bsvm:~$ touch datei
studi@bsvm:~$ mv datei foo
studi@bsvm:~$ ls
Bilder  Dokumente  Downloads  foo
Musik  Öffentlich  Schreibtisch  Videos  Vorlagen
studi@bsvm:~$ mv foo Downloads/
studi@bsvm:~$ ls Downloads
foo
```

Programmierung in C - 1

```c
#include <stdio.h>

int globalUninit;
int globalInit = 3;

int fib(int n) {
    if (n == 0) {return 0;}
    if (n == 1) {return 1;}
    return fib(n - 1) + fib(n - 2);
}
```

Programmierung in C - 2

```
int main(void) {
    int lokal = 3;

    printf("Adresse von globalUninit ist %p\n",
           (void*)&globalUninit);
    printf("Adresse von globalInit ist %p\n",
           (void*)&globalInit);
    printf("Adresse von lokal ist %p\n",
           (void*)&lokal);

    printf("Die 7. Fibonacci-Zahl lautet: %d\n", fib(7));

    return 0;
}
```

Programmieren in C

1. Probiert verschiedene Werte (mehrere Zehner bis hin zu mehreren Tausend) für den Parameter n aus:

a) Warum stimmt das Ergebnis ab einem bestimmten Wert für n nicht mehr? Wie groß ist n in eurem Fall?

b) Warum läuft das Programm ab einem bestimmten Wert nicht mehr bis zum Ende? Wie groß ist n in eurem Fall?

Programmieren in C

a) Warum stimmt das Ergebnis ab einem bestimmten Wert für n nicht mehr? Wie groß ist n in eurem Fall?

- **Bei n > 46**
  - Kein korrektes Ergebnis, da ein Integer-Überlauf stattfindet.

Programmieren in C

b) Warum läuft das Programm ab einem bestimmten Wert nicht mehr bis zum Ende? Wie groß ist n in eurem Fall?

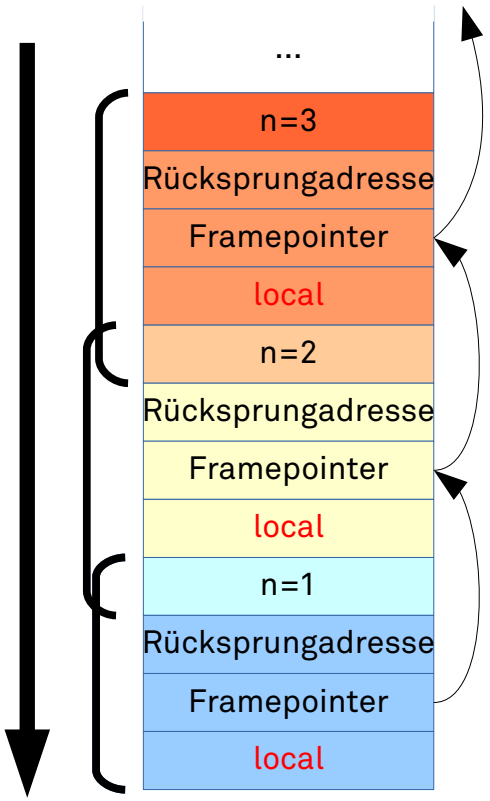
- **Bei n ≥ 45 dauert die Berechnung lange**
  - Bei großen n dauert die Berechnung exponentiell länger, da jede vorherige Fibonacci-Zahl vollständig berechnet werden muss.
- **Bei n ≥ 260000 stürzt das Programm ab**
  - Prozesse haben nur einen begrenzten Stack
  - Speicherzugriffe über die untere Grenze des Stacks hinaus verursachen *segmentation fault* → Prozess wird beendet
  - Zahl variiert, da der Kernel bei jeder Ausführung den Stackpointer zufällig initialisiert

2. Warum wird die Adresse einer lokalen Variablen in der rekursiven Funktion immer kleiner, wenn die Funktion immer weiter „rekursiv absteigt“?

2. Warum wird die Adresse einer lokalen Variablen in der rekursiven Funktion immer kleiner, wenn die Funktion immer weiter „rekursiv absteigt“?

- Bei **jedem Aufruf einer Funktion** wird ein **neuer *Stackframe*** auf dem Stack **abgelegt**
- Der Stack wächst dabei „nach unten”
- **Der *Stackframe*** enthält die **Parameter, die *Rücksprungadresse*, den *Framepointer* und auch die lokalen Variablen**

→ **Die Adresse lokaler Variablen** werden beim rekursiven Abstieg immer **kleiner**



3. Warum liegt eine globale int-Variable an einer völlig anderen Adresse?

3. Warum liegt eine globale int-Variable an einer völlig anderen Adresse?

Globale Variablen werden **nicht** auf dem **Stack**, sondern im **BSS-**(uninitialisiert) oder **Datensegment** (initialisiert) abgelegt.

Programmierung in C – Extended

```c
int main(int argc, char *argv[])
{
    if (argc < 2) {
        printf("Fehler: Keine Obergrenze angegeben\n");
        return -1;
    } else if (argc > 2) {
        printf("Fehler: Zu viele Parameter angegeben\n");
        return -2;
    }

    int input = atoi(argv[1]);
    if (input < 1) {
        printf("Fehler: Negative Zahl, 0 oder Text
                als Obergrenze angegeben\n");
        return -1;
    }

    printf("Die %d. Fibonacci-Zahl lautet: %d\n", input,
       fib(input));
    return 0;
}
```