

Verlässliche Systemsoftware

Übungen zur Vorlesung

Softwareentwurf

Phillip Raffeck, Simon Schuster, Peter Ulbrich

Technische Universität Dortmund
Lehrstuhl für Informatik 12 (Arbeitsgruppe Systemsoftware)
<https://sys.cs.tu-dortmund.de>

Wintersemester 2020



Anforderungen

- liest ASCII-Text über Standardeingabe ein
- zählt vorkommende Zeichen, Wörter und Zeilen
- Ausgabe: *<Anzahl Zeilen> <Anzahl Wörter> <Anzahl Zeichen>*



```
static int e,n,j,o,y;int main(){for(++o;(n=~getchar());e+=11==n,y++)  
o=n>0xe^012>n&&'`'^n^65?!n:!o?++j:o;printf("%8d%8d%8d\n",e^n,j+=!o&&y,y);}
```

- + erfüllt Anforderungen
- ✗ schwer zu lesen
- ✗ noch schwerer zu verstehen

Umsetzungsversuch 2

```
#include <stdio.h>
typedef size_t CharCountTy; typedef size_t WordCountTy; typedef size_t LineCountTy;
static void inc_char_count(CharCountTy *c) {*c += 1;}
static void inc_word_count(WordCountTy *w) {*w += 1;}
static void inc_line_count(LineCountTy *l) {*l += 1;}
typedef struct {int character; int error; int done;} ReadResTy;
static int isWordTerminator(int c) {return c == ' ' || (c >= '\t' && c <= '\r');}
static int isLineTerminator(int c) {return c == '\n';}
static ReadResTy getCharacter(void) { ReadResTy r;
    r.character = getchar(); r.done = 0; r.error = 0;
    if(r.character == EOF) {if(feof(stdin)) {r.done = 1;} else {r.error = 1;}} return r;
}
int main(void) {
    CharCountTy char_count = 0; WordCountTy word_count = 0;
    LineCountTy line_count = 0; int in_word = 0; ReadResTy input;
    while((input = getCharacter()), !input.error && !input.done) {
        inc_char_count(&char_count);
        if(isWordTerminator(input.character) && in_word) {
            inc_word_count(&word_count); in_word = 0;
        } else if(!isWordTerminator(input.character)) {in_word = 1;}
        if(isLineTerminator(input.character)) {inc_line_count(&line_count);}
    }
    if(input.error) {return -1;} // Something went wrong...
    printf("%8lu%8lu%8lu\n", line_count, word_count, char_count); return 0;
}
```

✗ vielleicht etwas zu viel des Guten ...



All problems in computer science can be solved by another level of indirection, except for the problem of too many layers of indirection.

—David J. Wheeler

- guter Softwareentwurf ist mehr Kunst als Wissenschaft
- keine Patentlösung



Modifizierbarkeit: lokale Veränderbarkeit

- ~ Änderungen an Anforderungen umsetzbar
- ~ Fehler korrigierbar

Effizienz: optimaler Betriebsmittelbedarf

- wird häufig zu früh berücksichtigt

Verlässlichkeit: lange Zeit funktionsfähig ohne menschlichen Eingriff

- gutmütiges Ausfallverhalten
- muss von Anfang an eingeplant sein!

Verständlichkeit: Isolierung von

- Daten
- Algorithmen



Abstraktion: wichtige Details hervorheben

Kapselung: unnötige Details verbergen

Einheitlichkeit: konsistente Notation

Vollständigkeit: alle wichtigen Aspekte berücksichtigt

Testbarkeit: muss von Anfang an eingeplant werden



Abstraktion: wichtige Details hervorheben

Kapselung: unnötige Details verbergen

Einheitlichkeit: konsistente Notation

Vollständigkeit: alle wichtigen Aspekte berücksichtigt

Testbarkeit: muss von Anfang an eingeplant werden

C macht es einem hier nicht leicht

~> **disziplinierte Herangehensweise** notwendig!

