Programmieren in C

Fabian Klötzl

MetaNooK 2017

Ich über mich

- Master in Informatik 2015
- PhD im MPI für Evolutionsbiologie, Plön
- 29 GitHub Repos
- Debian Med Team
- Folien unter `kloetzl.info#downloads`

Section 1

Eine Einführung in C

Hello World

```
#include <stdio.h>

int main(void)
{
    printf(”Hello,␣World!\n”);
    return 0;
}
```

Warum C im Jahre 2017?

Warum C im Jahre 2017?

Performance

Es gibt genügend gute andere Optionen: C++, D, Rust, Go.
Die bieten gleichzeitig bessere Abstraktionen, mehr Features, mehr Sicherheit, . . .

Warum C im Jahre 2017?

Performance

Es gibt genügend gute andere Optionen: C++, D, Rust, Go.
Die bieten gleichzeitig bessere Abstraktionen, mehr Features, mehr Sicherheit, . . .

Systems Programming

Kernel, Treiber, Glue-Code

Features

- Statisches Typsystem
- Strukturen
- Zeiger
- Compiliert schnell
- Minimal

Warum nicht C++?

```
auto numbers = std::vector<int>{0,1,2,3,4,5,6,7,8,9};

auto count = 0;
auto odd = [=] (int n) mutable {
    if (n & 1) {
        printf("%i␣%i\n", n, count);
        count++;
        return true;
    }
    return false;
};

std::remove_if(std::begin(numbers), std::end(numbers), odd);
```

C++ is a beast

Expected Output

```
1 0
3 1
5 2
7 3
9 4
```

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=81482

C++ is a beast

Expected Output

```
1 0
3 1
5 2
7 3
9 4
```

Actual Output

```
1 0
3 0
5 1
7 2
9 3
```

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=81482

Wie groß ist ein int?

a) 16 bit
b) 32 bit
c) 40 bit
d) 64 bit

Wie groß ist ein int?

a) 16 bit
b) 32 bit
c) 40 bit
d) 64 bit

Lösung

Alles ist möglich!
2 byte $\leq$ short $\leq$ int,
int $\leq$ long,
4 byte $\leq$ long

Vorzeichen

Signed oder Unsigned, das ist hier die Frage.

- char
- short
- int
- size_t

Vorzeichen

Signed oder Unsigned, das ist hier die Frage.

- char
- short
- int
- size_t

Lösung

- char: je nach Encoding
- short: signed
- int: signed
- size_t: unsigned (ssize_t)

Alle Typen die ihr je braucht

- char: Byte, Zeichen
- int: Bool, File Descriptor
- size_t: Indices, Größen
- double: Reelle Zahlen
- *arrays*: Arrays
- struct: Zusammengesetzte Datentypen
- *pointer*: Zeiger

Funktionen

```c
int add(int a, int b);
void fn(void);

int add(int a, int b)
{
	return a + b;
}
```

Strukturen

```c
struct range {
    size_t lower, upper;
    double weight;
};

struct range r = {0, 100, 0.5};
double area = (r.upper - r.lower) * r.weight;
struct range s = r ; // copy
```

Statische Arrays

```c
int buffer[1024] = {0};

buffer[0] = 23;
memset(buffer, 1, sizeof(buffer));
printf("%i\n", buffer[0]); // 16843009 = 0x1010101
```

```c
void fn(size_t count)
{
    int arr[count] = {0};

    do_something(arr);

    // automatic deallocation
}
```

Zeiger

Schuhe

Zeiger

```c
struct range normalize(struct range r) {
	double inv = 1.0 / (r.upper - r.lower);
	r.weight = inv;
	return r;
}

void normalize2(struct range *range_ptr) {
	double inv = 1.0 / (( *range_ptr).upper - range_ptr->lower);
	range_ptr->weight = inv;
}

struct range rr;
rr = normalize(rr);
normalize2(&rr);
```

Zeiger

NULL

```
void normalize2(struct range *range_ptr) {
    double inv = 1.0 / (( *range_ptr).upper - range_ptr->lower);
    range_ptr->weight = inv;
}

normalize2(NULL); // SEGFAULT
int foo;
normalize2(&foo); // int* != struct range*
```

Zeiger

```
void swap_int(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int x = 4;
int y = 5;
swap_int(&x, &y);
```

Strings

Definition

Ein *String* ist eine Folge von Zeichen, die mit NUL (einem Null-Byte) endet.

```
const char *str = "Hallo";
// str[0] = 'H'
// str[1] = 'a'
// str[2] = 'l'
// str[3] = 'l'
// str[4] = 'o'
// str[5] = '\0'
strlen(str) // 5
printf("%s\n", str) // Hallo
```

Strings

```
const char *str = "Hallo";
str[1] = 'e';
str[4] = '\0';
strlen(str) // 4
printf("%s\n", str) // Hell
```

Strings

strdup

```
const char *str = "Hallo";
char *mut = strdup(str);
mut[1] = 'e';
mut[4] = '\0';
strlen(str) // 5
strlen(mut) // 4
```

Speicherverwaltung

malloc, free

```
int *arr = malloc(count * sizeof(int));

do_something(arr);

free(arr);
```

Daumenregel

grep −c 'free' ≈ grep −c 'malloc'

gilt auch für strdup!

Speicherverwaltung

err

```c
int *arr = malloc(count * sizeof(int));
if (arr == NULL) {
	err(errno, "out of memory");
}

do_something(arr);

free(arr);
```

Speicherverwaltung

reallocarray

```
int *arr = reallocarray(NULL, count, sizeof( *arr));
if (arr == NULL) {
	err(errno, "out␣of␣memory");
}

do_something(arr);

free(arr);
```

Section 2

Übung

Vektor
std::vector

Eigenschaften:

- kontinuierlicher Speicher
- dynamisch verlängerbar

Operationen:

- Erstellen
- Ein Element anhängen
- Anzahl der gespeicherten Elemente abfragen
- Auf Daten zugreifen
- Speicher freigeben

Elementtyp: long

Vektor

Struktur

```
struct vector_long {
	long *data;
	size_t size;
	size_t capacity;
};
```

Vektor
Operationen

```c
// int vector_long_init(struct vector_long*);
void vector_long_free(struct vector_long*);
int vector_long_push(struct vector_long*, long);
size_t vector_long_size(struct vector_long*);
long* vector_long_data(struct vector_long*);
```

Vektor

init

```c
int vector_long_init(struct vector_long* vec) {
	if (!vec) return 1;

	vec->data = reallocarray(NULL, 4, sizeof(*vec->data));
	vec->size = 0;
	vec->capacity = 4;

	// return 0 iff successful
	return vec->data == NULL ? 1 : 0;
}
```

Vektor

free

```c
void vector_long_free(struct vector_long* vec) {
	if (!vec) return;
	free(vec->data);
	*vec = (struct vector_long){0};
}
```

Vektor

data, size

```c
long *vector_long_data(struct vector_long* vec) {
	if (!vec) return NULL;
	return vec->data;
}

size_t vector_long_size(struct vector_long *vec) {
	if (!vec) return 0;
	return vec->size;
}
```

Vektor

push

```
int vector_long_push(struct vector_long* vec, long element) {
    if (!vec) return 1;

    if (vec->size < vec->capacity) {
        vec->data[vec->size++] = element;
        return 0;
    }

    long *ptr = reallocarray(vec->data, vec->size / 2, 3 * sizeof(long));
    if (!ptr) return 1;
    vec->data = ptr;
    vec->capacity = (vec->capacity / 2) * 3;
    vec->data[vec->size++] = element;

    return 0;
}
```

Section 3

Tipps und Tools

Commandline Arguments

```
int main(int argc, char **argv) {
    int c;
    int flag = 0;
    int iterations = 10;
    while((c = getopt(argc, argv, "hfi:")) != −1) {
        if (c == 'h') usage(EXIT_SUCCESS);
        if (c == 'f') flag = 1;
        if (c == 'i') iterations = atoi(optarg);
        if (c == '?') usage(EXIT_FAILURE);
    }
    ...
}
```

Konvertierung

string to int

```c
int iterations;
const char *errstr;

iterations = strtonum(optarg, 1, 64, &errstr);
if (errstr)
        errx(1, "number␣of␣iterations␣is␣%s:␣%s", errstr, optarg);
```

Konvertierung

int to string

```
int iterations = 10;
char *str;

int check = asprintf(&str, "%i", iterations);
if (check < 0)
        errx(1, "critical␣error");
```

Man Pages

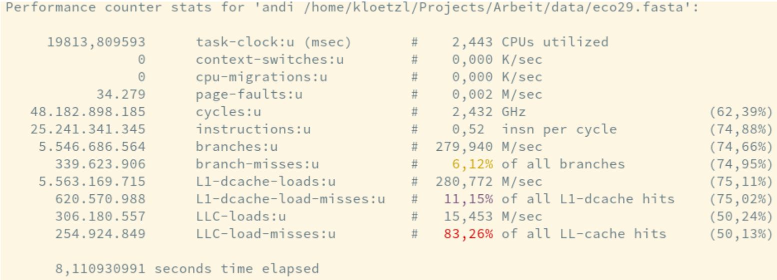
- man malloc
- man 3 free
- man ascii
- help2man

Compilieren

- Compiler: clang, gcc
- Warnings: -Wall -Wextra
- Optimierung: -O0, -O2, -O3
- -march=native
- Debugging: -ggdb -fno-omit-frame-pointer

perf stat

perf stat −d tool file.ext



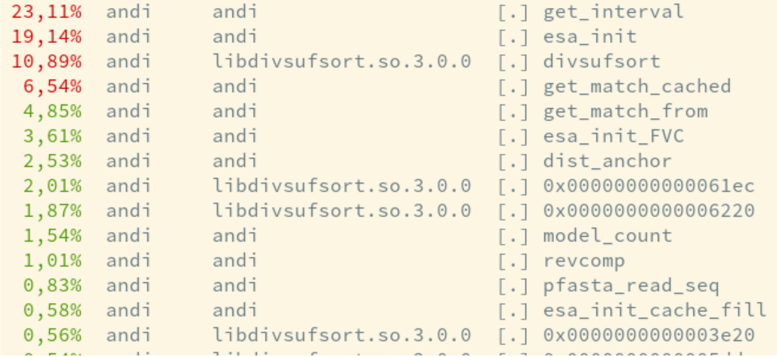
```
Performance counter stats for 'andi /home/kloetzl/Projects/Arbeit/data/eco29.fasta':

    19813,809593      task-clock:u (msec)       #    2,443 CPUs utilized
               0      context-switches:u        #    0,000 K/sec
               0      cpu-migrations:u          #    0,000 K/sec
          34.279      page-faults:u             #    0,002 M/sec
  48.182.898.185      cycles:u                  #    2,432 GHz                      (62,39%)
  25.241.341.345      instructions:u            #    0,52  insn per cycle           (74,88%)
   5.546.686.564      branches:u                #  279,940 M/sec                    (74,66%)
     339.623.906      branch-misses:u           #    6,12% of all branches          (74,95%)
   5.563.169.715      L1-dcache-loads:u         #  280,772 M/sec                    (75,11%)
     620.570.988      L1-dcache-load-misses:u   #   11,15% of all L1-dcache hits    (75,02%)
     306.180.557      LLC-loads:u               #   15,453 M/sec                    (50,24%)
     254.924.849      LLC-load-misses:u         #   83,26% of all LL-cache hits     (50,13%)

     8,110930991 seconds time elapsed
```

perf record

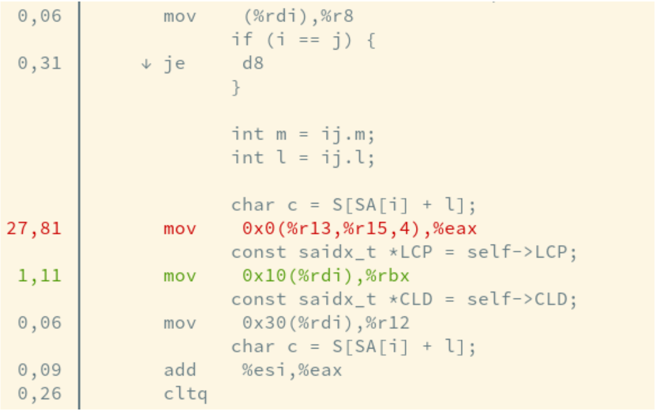
perf record tool file.ext
perf report



```
23,11%  andi    andi                   [.] get_interval
19,14%  andi    andi                   [.] esa_init
10,89%  andi    libdivsufsort.so.3.0.0 [.] divsufsort
 6,54%  andi    andi                   [.] get_match_cached
 4,85%  andi    andi                   [.] get_match_from
 3,61%  andi    andi                   [.] esa_init_FVC
 2,53%  andi    andi                   [.] dist_anchor
 2,01%  andi    libdivsufsort.so.3.0.0 [.] 0x00000000000061ec
 1,87%  andi    libdivsufsort.so.3.0.0 [.] 0x0000000000006220
 1,54%  andi    andi                   [.] model_count
 1,01%  andi    andi                   [.] revcomp
 0,83%  andi    andi                   [.] pfasta_read_seq
 0,58%  andi    andi                   [.] esa_init_cache_fill
 0,56%  andi    libdivsufsort.so.3.0.0 [.] 0x0000000000003e20
```

perf record

perf record tool file.ext
perf report



```
 0,06 │        mov    (%rdi),%r8
      │               if (i == j) {
 0,31 │      ↓ je      d8
      │               }
      │
      │               int m = ij.m;
      │               int l = ij.l;
      │
      │               char c = S[SA[i] + l];
27,81 │        mov     0x0(%r13,%r15,4),%eax
      │               const saidx_t *LCP = self->LCP;
 1,11 │        mov     0x10(%rdi),%rbx
      │               const saidx_t *CLD = self->CLD;
 0,06 │        mov     0x30(%rdi),%r12
      │               char c = S[SA[i] + l];
 0,09 │        add     %esi,%eax
 0,26 │        cltq
```

Bad Code

```
name[N] = (char*) calloc(kseq->name.l + 1, sizeof(char));
strcpy(name[N], kseq->name.s);
```

https://github.com/tseemann/snp-dists

Bad Code

```
name[N] = (char*) calloc(kseq->name.l + 1, sizeof(char));
strcpy(name[N], kseq->name.s);
```

https://github.com/tseemann/snp-dists

1. sizeof(char) ist immer 1
2. warum *calloc*, wenn wir gleich wieder überschreiben?
3. *malloc* nicht casten
4. wir kennen die Länge → *memcpy*
5. memory leak (not shown)

Bad Code

Verbessert

```c
name[N] = (char*) calloc(kseq->name.l + 1, sizeof(char));
strcpy(name[N], kseq->name.s);
```

```c
name[N] = malloc(kseq->name.l + 1);
memcpy(name[N], kseq->name.s, kseq->name.l + 1);
```

```c
name[N] = strdup(kseq->name.s);
```

Bad Code 2

```
int ** array;
array = malloc(nrows * sizeof(double * ));

for(i = 0; i < nrows; i++){
	array[i] = malloc(ncolumns * sizeof(double));
}

array[0][1] = 3.141;
```

http://moreisdifferent.com/2015/07/16/why-physicsts-still-use-fortran/

Bad Code 2

```
double **array = malloc(nrows * sizeof(double *));

for(i = 0; i < nrows; i++){
        array[i] = malloc(ncolumns * sizeof(double));
}

array[0][1] = 3.141;
```

Bad Code 2

```
double **array = malloc(nrows * sizeof(double *));
double *base = malloc(ncolumns * nrows * sizeof(double));

for(i = 0; i < nrows; i++){
        array[i] = base + i * ncolumns;
}

array[0][1] = 3.141;
```

Bad Code 2

```
double *base = malloc(ncolumns * nrows * sizeof( *base));

#define B(X,Y) (base[(X)*ncolumns + (Y)])

B(0,1) = 3.141;
```

Guter Code

- BSD Userland
- libc
- Debian Code Search
- GitHub

Optionale und Benamte Argumente

```
struct opts { size_t size; };

int vector_long_init(struct vector_long* vec, struct opts o) {
    vec->data = malloc(o.size * sizeof(long));
    ...
}

#define vector_long_init(VEC, ...) \
 vector_long_init((VEC), (struct opts){.size = 4, __VA_ARGS__})


vector_long_init(&vec);
vector_long_init(&vec, .size = 10);
```

Bad Code 3

```c
char *path_name(const struct name_path *path, const char *name)
{
    const struct name_path *p;
    int nlen = strlen(name);
    int len = nlen + 1;

    for (p = path; p; p = p->up) {
        if (p->elem_len)
            len += p->elem_len + 1;
    }
    char *n = xmalloc(len);
    ...
    return n;
}
```

https://github.com/git/git/blob/v1.7.0/revision.c

Bad Code 3

Verbessert

```c
char *path_name(const struct name_path *path, const char *name)
{
    const struct name_path *p;
    size_t nlen = strlen(name);
    size_t len = nlen + 1;

    for (p = path; p; p = p->up) {
        if (p->elem_len)
            len += p->elem_len + 1;
    }
    char *n = xmalloc(len);
    ...
    return n;
}
```

Bad Code 4

```
char *line = fd_read_line (fd);
char *endl;
if (line == NULL)
    break;

remaining_chunk_size = strtol (line, &endl, 16);
xfree (line);

if (remaining_chunk_size == 0) {
        line = fd_read_line (fd);
        xfree (line);
        break;
}
fd_read (fd, dlbuf, remaining_chunk_size, -1);
```

https://access.redhat.com/security/cve/cve-2017-13089

Bad Code 4
Verbessert

```
char *line = fd_read_line (fd);
char *endl;
if (line == NULL)
  break;

remaining_chunk_size = strtol (line, &endl, 16);
xfree (line);

if (remaining_chunk_size < 0)
    return false;

if (remaining_chunk_size == 0) {
    line = fd_read_line (fd);
    xfree (line);
    break;
}
fd_read (fd, dlbuf, remaining_chunk_size, -1);
```

Objektorientierung

```
struct range {
    size_t lower, upper;
    double weight;
};

struct colored_range {
    size_t lower, upper;
    double weight;
    int color;
};
```

Objektorientierung

```
struct colored_range {
        struct range base;
        int color;
};

struct colored_range cr;
cr.base.left = 0;
normalize(&cr.base);
```

Objektorientierung

```
struct colored_range {
    struct range; // gcc −fms−extensions
    int color;
};

struct colored_range cr;
cr.left = 0;
normalize(&cr); // gcc −fplan9−extensions
```

Objektorientierung

```c
struct colored_range {
    union {
        struct range;
        struct range as_range;
    };
    int color;
};

struct colored_range cr;
cr.left = 0;
normalize(&cr.as_range);
```

Bad Code 5

```
struct Score
{
  long int value;
  char used;
};

struct Node
{
  Score S[3];
  int m_max;
};

struct Node huge_vector[1000]; // conceptually
```

Bad Code 5

Verbessert

```
struct Node
{
  long int values[3];
  char used[3];
  int m_max;
};

struct Node huge_vector[1000]; // conceptually
```

Debugger

- *gdb −−args* !!
- *run* Starten
- *list* Aktuelle Codezeile ausgeben
- *p* Variable ausgeben

Sanitizer

- *−fsanitize=address*
- Address: Erkennt Pointerfehler
- Integer: Erkennt Over/Underflows
- Undefined: UB
- Thread: Race-Conditions
- Dynamisch (zur Laufzeit), gute Unittests!

Statische Analyse

- *scan−build gcc*
- *scan−build make*
- Findet Bugs wzB. Null-Pointer-Deref
- Gibt Pfad zum Bug an.
- Bei Compilierung (statisch)

Formatieren

- *clang−format −i ∗.c*
- Formatiert euren Code automatisch passend zu (euren Vorlieben) den Vorgaben des Projektes
- Braces, Zeilenumbrüche, Includes sortieren, . . .
- Clang-Format-Datei:

```
BasedOnStyle: LLVM
IndentWidth: 4
TabWidth: 4
UseTab: Always
AllowShortIfStatementsOnASingleLine: true
AllowShortFunctionsOnASingleLine: false
```

Instruktionssets

```c
int func(char *bar) __attribute__((target_clones("AVX,default")))
{
    // code goes here
    return 0;
}
```

1. Dynamische Auflösung zur Laufzeit
2. Gute Lösung für Paketverwaltung

Laufzeitmessen

- $/usr/bin/time -f$ "%e␣%M␣%P" *tool args*
- $\%e$: elapsed time (wall clock)
- $\%M$: memory peak
- $\%P$: durchschnittliche CPU-Auslastung
- $\hat{t} = t + t_{clock} + t_{sched} + t_{caching}$
- $\hat{t}_1, \hat{t}_2, \hat{t}_3 \rightarrow t?$

Laufzeitmessen

- $/usr/bin/time -f$"%e␣%M␣%P"$tool\ args$
- %e: elapsed time (wall clock)
- %M: memory peak
- %P: durchschnittliche CPU-Auslastung
- $\hat{t} = t + t_{clock} + t_{sched} + t_{caching}$
- $\hat{t}_1, \hat{t}_2, \hat{t}_3 \to t?$
- $t_x \geq 0$
- $t = \min(\hat{t}_1, \hat{t}_2, \hat{t}_3)$

Wofür ich keine Zeite hatte

- unions, enums
- undefined behaviour → CppCon Talks
  unsigned long $file\_size = 5 << 30; //5gb$
- const, inline
- typedef
- valgrind
- Google Benchmark
- `https://matt.sh/howto-c`
- fuzzing
- strict aliasing

Zusammenfassung

Zusammenfassung

kloetzl.info#downloads