Verlässliche Echtzeitsysteme - Übungen

Filter

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Überblick

1. C-Quiz Teil I
2. Festkommaarithmetik
3. Softwareentwurf
4. Hinweise zur Aufgabe: Filter

Annahmen

- C99
- x86 bzw. x86-64, d. h.
  - vorzeichenbehaftete Integer als Zweierkomplement implementiert
  - `char` hat 8 Bit
  - `short` hat 16 Bit
  - `int` hat 32 Bit
  - `long` hat 32 Bit auf x86 und 64 Bit auf x86-64

Frage 3 [2]

Angenommen: `int x = 1;` **Zu was wird** `(unsigned short)x > -1` **ausgewertet?**

1. `0`
2. 1
3. nicht definiert

Erklärung

- vor dem Vergleich beide Operanden nach `int` umgewandelt
- weil dies ohne Wertverlust geschehen kann

⇝ hier werden zwei `signed`-Werte verglichen

⇝ ein `unsigned int` würde nicht umgewandelt werden!

Frage 4 [2]

Zu was wird `-1L > 1U` auf x86-64 ausgewertet? Auf x86?

1. beides `0`
2. beides `1`
3. `0` auf x86-64, `1` auf x86
4. `1` auf x86-64, `0` auf x86

Erklärung

- auf x86-64 ist `int` kürzer als `long`

⇝ `unsigned int` wird zu `long` ⇝ `-1L > 1L` ⇒ `0`

- auf x86 entspricht `int` dem Datentyp `long`

⇝ `UINT_MAX > 1U` ⇒ `1`

Frage 5 [2]

Zu was wird `SCHAR_MAX == CHAR_MAX` ausgewertet?

1. `0`
2. `1`
3. implementation defined

Erklärung

- C99 schreibt nicht vor ob `char` vorzeichenbehaftet ist
- auf x86 und x86-64 ist `char` für gewöhnlich vorzeichenbehaftet

Frage 6 [2]

Zu was wird `UINT_MAX + 1` ausgewertet?

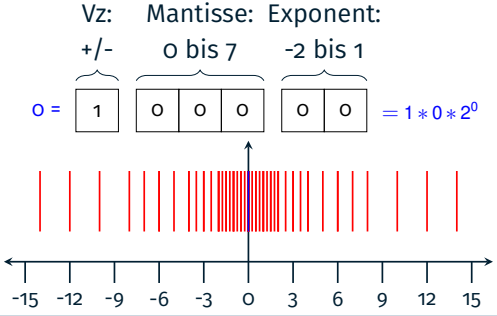
1. `0`
2. `1`
3. `INT_MIN`
4. `UINT_MIN`
5. nicht definiert

Erklärung

Der C-Standard garantiert, dass `UINT_MAX + 1 == 0`

Inhalt

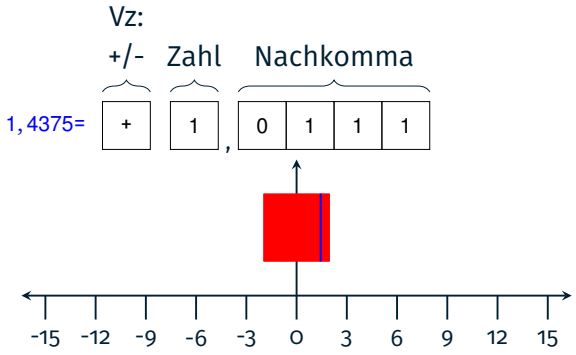
Fließkommazahlen



IEEE 754

- Noch komplexer:
  - normalisierte/denormalisierte Darstellung
  - Rundung, Fehlersemantik, …
  - NaN, ∞, …
- `https://ieeexplore.ieee.org/document/4610935/`

Festkommazahlen: Grundlagen



C-Standard und Zahlendarstellung

- Zahlendarstellung im Standard nicht festgelegt:
  - Einerkomplement
  - Vorzeichen und Magnitude
  - Zweierkomplement
- Heute meist Zweierkomplement $\rightsquigarrow$ kein dediziertes Vorzeichenbit

Festkommaarithmethik – Motivation

```
float func(void){
  volatile float a = 23.42;
  volatile float b = 12.34;
  return a * b;
}
```

```
func:
  push  {r7, lr}
  sub   sp, #8
  add   r7, sp, #0
  ldr   r3, [pc, #28] ; float a
  str   r3, [r7, #4]
  ldr   r3, [pc, #28] ; float b
  str   r3, [r7, #0]
  ldr   r2, [r7, #4]
  ldr   r3, [r7, #0]
  adds  r0, r2, #0 ; Param 1
  adds  r1, r3, #0 ; Param 2
  bl    3a6c <__aeabi_fmul>
  adds  r3, r0, #0
  adds  r0, r3, #0
  mov   sp, r7
  add   sp, #8
  pop   {r7, pc}
```

- Setup
  - Plattform: ARM Cortex-M0+
  - Compiler: arm-gcc
- Funktion `__aeabi_fmul` : 300 Zeilen Assembler
- Keine Fließkommaeinheit (engl. floating-point unit, FPU) vorhanden
- Emulation der **Fließkommaarithmetik in Software**

Festkommaarithmetik – Q-Notation

- Mikrocontroller ohne *Fließkommaeinheit*
- *Kein EAN* für Fließkommazahlen
  - $\rightsquigarrow$ *Festkommaarithmethik* mit Ganzzahlen
- Zahlenformat häufig in Q-Notation [1] angegeben
- *Qm.n* $\rightsquigarrow$ Festkommazahl mit
  - m Bit vor dem Komma, n nach dem Komma, ein Vorzeichenbit
  - Wertebereich: $[-2^m, 2^m - 2^{-n}]$
  - Auflösung: 2^{-n}
- Implementierung für Übungsaufgabe *vorgegeben*

Implementierung als Integer

$\rightsquigarrow$ passendes Q-Format ist **anwendungsspezifisch**

Q-Notation – Beziehung zu Fließkommazahlen

von Fließkomma nach Qm.n

1. Multiplikation mit 2^n
2. Runden auf die nächste Ganzzahl

von Qm.n nach Fließkomma

1. Umwandlung in Fließkommazahl $\rightsquigarrow$ cast
2. Multiplikation mit 2^{-n}

Operationen – Addition/Subtraktion

- Addition und Subtraktion wie bei Ganzzahlen

Addition

```
int32_t       a = ...;
int32_t       b = ...;
int32_t result = a + b;
```

```
    2 , 8 0
+ 1 3 , 3 7
  (1)
= 1 6 , 1 7
```

Subtraktion

```
int32_t       a = ...;
int32_t       b = ...;
int32_t result = a - b;
```

```
  1 6 , 1 7
-   2 , 8 0
  (1)
= 1 3 , 3 7
```

Operationen – Multiplikation/Division

- Braucht Zwischenergebnis von doppelter Bitbreite

Multiplikation

```
#define K   (1 << (n - 1))
int32_t     a = ...;
int32_t     b = ...;
int64_t temp = (int64_t) a * (int64_t) b;
        temp += K;
int32_t result= temp >> n;
```

$$
\begin{aligned}
& a \cdot b \\
\overset{Q.n}{=} & (a \cdot 10^n) \cdot (b \cdot 10^n) \\
= & (a \cdot b) \cdot 10^{2n} \\
\neq & (a \cdot b) \cdot 10^n
\end{aligned}
$$

Division

```
int32_t      a = ...;
int32_t      b = ...;
int64_t   temp = (int64_t) a << n;
          temp += b / 2;
int32_t result = temp / b;
```

$$
\begin{aligned}
& \frac{a}{b} \overset{Q.n}{=} \frac{a \cdot 10^n}{b \cdot 10^n} \\
= & \frac{a}{b} \neq \frac{a}{b} \cdot 10^n
\end{aligned}
$$

- Siehe Implementierung in `fixedpoint.c`
- Vorsicht: Rundungsfehler durch Transformationen

Inhalt

Spezifikation: Simples Textstatistikprogramm

Anforderungen

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

Umsetzungsversuch 1

```
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

https://www.ioccc.org/2019/burton/prog.clean.c

Umsetzungsversuch 2

```c
#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 …

Gelungener Softwareentwurf

> *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

Ziele des Softwareentwurfs

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

Prinzipien des Softwareentwurfs

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

$\rightsquigarrow$ **disziplinierte Herangehensweise** notwendig!

Inhalt

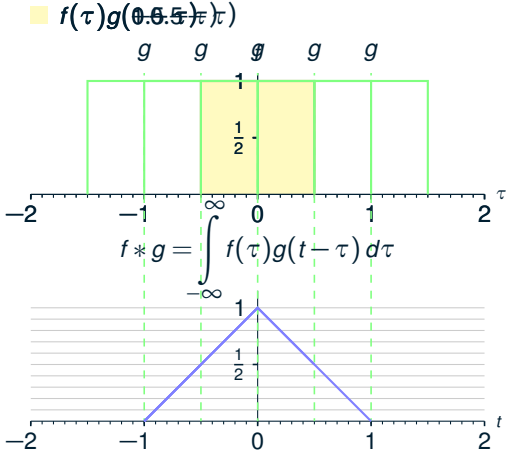
Hinweise zur Aufgabe Implementierung Filter

- Objekte identifizieren (z.B. Eingaben)
- Implementierung der Filterung durch **Faltung** (engl. convolution) mit Impulsantwort
  - f Signalwerte, g Filterwerte

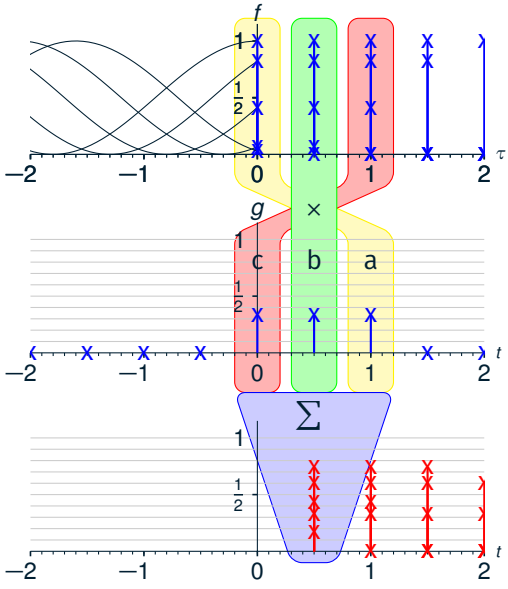
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m] \tag{1}$$

- Zunächst Verwendung von `float`, anschließend **Festkommaformat**

Beispiel: Faltung



Beispiel: Diskrete Faltung



$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m]$$

Exkurs: Prüfsummen

- Verkürzte Repräsentation eines Datensatzes (Bsp.: Git Commithash)
- Aus Eingabedaten errechnet
- Nutzen: Überprüfung der Datenintegrität
- Anforderungen
  - Stabil
  - Effizient berechenbar
  - Zuverlässige Fehlererkennung
- Beispiel: Quersumme mit Zehner-Restklasse:

$$\mathrm{CHECK}(12345) = (1+2+3+4+5) \mod 10 = 5$$

$$\mathrm{CHECK}(12355) = (1+2+3+5+5) \mod 10 = 6$$

  $\rightsquigarrow$ Fehlermodell: Schützt vor allen Einzifferfehlern

  $\rightsquigarrow$ jedoch bspw. kein Schutz gegen Vertauschung:

$$\mathrm{CHECK}(12354) = (1+2+3+5+4) \mod 10 = 5$$

- Wichtig: Fehlermodell, welche Arten von Bitfehlern werden erkannt?

Hinweise zur Aufgabe

- **Einzelnen Filterschritt** implementieren (kein Burst-Filter)
- Verwendung von **Q-Notation**
- *Wichtig für die späteren Experimente*
  1. Berechnungsdauer möglichst kurz (Fehlerinjektion dauert sehr lange)
  2. wollen keine verpflichtende Nutzung von `floats`
  3. Später dann: Keine Verwendung von `printf` bei Fehlerinjektion
  4. Konzeption einer einfachen Prüfsumme
- Aspekte:
  - Nutzung abstrakter Schnittstellen
  - Einfluss von Schnittstellen auf Verlässlichkeit
  - Entwurfsentscheidungen und -abwägung in der Systemimplementierung

Literatur (1)


E. L. Oberstar.
Fixed-point representation & fractional math.
Technical report, Oberstar Consulting, August 2007.


J. Regher.
A quiz about integers in c.