

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Florian Franzmann, Martin Hoffmann, Tobias Klaus

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
<http://www4.cs.fau.de>

26. Juni 2014



Umgang mit Ganzzahlfehlern

- C bietet viele subtile Fehlermöglichkeiten
 - Im C-Quiz haben wir einige kennengelernt
 - Was uns noch fehlt:
 - *Wie verhält man sich als Programmierer richtig?*
- ~> Heute ein paar Beispiele



Addition

Was soll da schon schiefgehen...

```
1 unsigned int func(unsigned int a, unsigned int b) {
2     return a + b;
3 }
```

Vorbedingungstest

```
1 #include <limits.h>
2 unsigned int func(unsigned int a, unsigned int b) {
3     if (UINT_MAX - a < b) { raise("wraparound"); }
4     return a + b;
5 }
```

Nachbedingungstest

```
1 unsigned int func(unsigned int a, unsigned int b) {
2     unsigned int ret = a + b;
3     if (ret < a) { raise("wraparound"); }
4     return ret;
5 }
```



Subtraktion

Was soll da schon schiefgehen...

```
1 unsigned int func(unsigned int a, unsigned int b) {
2     return a - b;
3 }
```

Vorbedingungstest

```
1 unsigned int func(unsigned int a, unsigned int b) {
2     if (a < b) { raise("wraparound"); }
3     return a - b;
4 }
```

Nachbedingungstest

```
1 unsigned int func(unsigned int a, unsigned int b) {
2     unsigned int ret = a - b;
3     if (ret > a) { raise("wraparound"); }
4     return ret;
5 }
```



Multiplikation

Was soll da schon schiefgehen...

```
1 unsigned int func(unsigned int a, unsigned int b) {
2     return a * b;
3 }
```

Vorbedingungstest

```
1 #include <limits.h>
2 unsigned int func(unsigned int a, unsigned int b) {
3     if (UINT_MAX / a < b) { raise("wraparound"); }
4     return a * b;
5 }
```



Explizite Typumwandlung

Was soll da schon schiefgehen...

```
1 unsigned int func(signed int a) {
2     return (unsigned int) a; /* keine Compilerwarning wg. Cast */
3 }
```

Vorbedingungstest

```
1 unsigned int func(signed int a) {
2     if (a < 0) { raise("wraparound"); }
3     return (unsigned int) a;
4 }
```



Explizite Typumwandlung

Was soll da schon schiefgehen...

```
1 unsigned char func(unsigned long int a) {
2     return (unsigned char) a; /* keine Compilerwarning wg. Cast */
3 }
```

Vorbedingungstest

```
1 unsigned char func(unsigned long int a) {
2     if (a > UCHAR_MAX) { raise("overflow"); }
3     return (unsigned char) a; /* keine Compilerwarning wg. Cast */
4 }
```



Explizite Typumwandlung

Was soll da schon schiefgehen...

```
1 signed char func(unsigned long int a) {
2     return (signed char) a; /* keine Compilerwarning wg. Cast */
3 }
```

Vorbedingungstest

```
1 #include <limits.h>
2 signed char func(unsigned long int a) {
3     if (a > SCHAR_MAX) { raise("overflow"); }
4     return (signed char) a;
5 }
```



Explizite Typumwandlung

Was soll da schon schiefgehen...

```
1 signed char func(signed long int a) {
2     return (signed char) a; /* keine Compilerwarning wg. Cast */
3 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed char func(signed long int a) {
4     if (a < SCHAR_MIN or SCHAR_MAX < a) { raise("overflow"); }
5     return (signed char) a; /* keine Compilerwarning wg. Cast */
6 }
```



Addition

Was soll da schon schiefgehen...

```
1 signed int func(signed int a, signed int b) {
2     return a + b;
3 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed int func(signed int a, signed int b) {
4     if ((b > 0 and a > INT_MAX - b)
5         or (b < 0 and a < (INT_MIN - b))) { raise("overflow"); }
6     return a + b;
7 }
```



Subtraktion

Was soll da schon schiefgehen...

```
1 signed int func(signed int a, signed int b) {
2     return a - b;
3 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed int func(signed int a, signed int b) {
4     if ((b > 0 and a < INT_MIN + b)
5         or (b < 0 and a > INT_MAX + b)) { raise("overflow"); }
6     return a - b;
7 }
```



Division

Was soll da schon schiefgehen...

```
1 signed long func(signed long a, signed long b) {
2     return a / b;
3 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed long func(signed long a, signed long b) {
4     if (a == LONG_MIN and b == -1) { raise("overflow"); }
5     return a / b;
6 }
```



Division

■ Reicht das schon?

Was soll da schon schiefgehen...

```
1 signed long func(signed long a, signed long b) {
2     if (a == LONG_MIN and b == -1) { raise("overflow"); }
3     return a / b;
4 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed long func(signed long a, signed long b) {
4     if (b == 0) { raise("division by zero"); }
5     if (a == LONG_MIN and b == -1) { raise("overflow"); }
6     return a / b;
7 }
```



Modulo

Was soll da schon schiefgehen...

```
1 signed long func(signed long a, signed long b) {
2     return a % b;
3 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed long func(signed long a, signed long b) {
4     if (b == 0) { raise("division by zero"); }
5     if (a == LONG_MIN and b == -1) { raise("overflow"); }
6     return a % b;
7 }
```



Negation

Was soll da schon schiefgehen...

```
1 signed long func(signed long a) {
2     return -a;
3 }
```

Vorbedingungstest

```
1 #include <limits.h>
2 signed long func(signed long a) {
3     if (a == LONG_MIN) { raise("overflow"); }
4     return -a;
5 }
```



Multiplikation

Was soll da schon schiefgehen...

```
1 signed int func(signed int a, signed int b) {
2     return a * b;
3 }
```

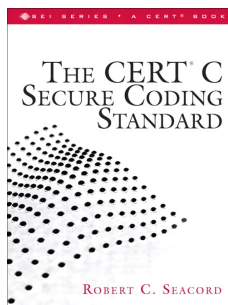
Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed int func(signed int a, signed int b) {
4     if (a > 0 and b > 0 and a > INT_MAX / b) { raise("overflow"); }
5     if (a > 0 and b < 0 and b < INT_MIN / a) { raise("overflow"); }
6     if (a < 0 and b > 0 and a < INT_MIN / b) { raise("overflow"); }
7     if (a < 0 and b < 0 and b < INT_MAX / a) { raise("overflow"); }
8     return a * b;
9 }
```



Das ist leider noch nicht alles . . .

- Es gibt noch mehr Sachen, auf die man aufpassen muss
- Literaturempfehlung:



- auch online:

<https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Coding+Standard>



Table of Contents

1 Abfangen von Integer-Fehlern

2 Es geht auch schöner



Wunschliste I

Was wünschen wir uns von einer Programmiersprache?

- Explizitheit
 - so viel formale Information wie möglich im Code
 - so viel formale Information wie möglich in der Spezifikation
- Keine Mogeleyen, Hacks, . . .
 - Möglichst keine Zeiger
 - Keine Mehrdeutigkeit
 - Typumwandlung nur explizit
- Zweifelhafte Dinge *möglich* machen
 - ↪ aber nicht versteckt!



Wunschliste II

Was noch?

- Gute Lesbarkeit
 - ↪ Hochsprache
 - Kopieren von Arrays durch Zuweisung, nicht memcpy
 - Kopieren von Teilarrays
- Sicherheit
 - Wertüberprüfung zur Laufzeit
 - Automatische Prüfung auf Pufferüberlauf
 - Parametermodi (rein, raus, rein/raus)
 - Verträge, auch für Typen
- Abstraktion, Kapselung ↪ Namensräume, Module
- Explizit vereinbare Datenrepräsentation
- . . .



Die Programmiersprache Ada

- Erfüllt alle unsere Anforderungen
- Allgemein anwendbare Programmiersprache
- *Besonderes Augenmerk auf Safety*
- Keine akademische „Spinnerei“ ...
 - Airbus A320, A330, A340, A380
 - Boeing 737, 747, 757, 767, 777, 787
 - TGV
 - Metro Linie 14 in Paris ~ selbstfahrend
 - INMARSAT, NASA Cloudsat, European Space Agency Infrared Space Telescope
 - ...

C wurde als Kandidat für Ada in Erwägung gezogen ...

Die Väter von C lehnten das jedoch ab!



Wertbeschränkte Typen

- In Ada erben von jedem Typ möglich
- Auch von Ganzzahl-, Gleitkomma- und Festkommatypen
- *Beschränkung des Wertebereichs* möglich
- Verletzung der Beschränkung führt zumindest zu Laufzeitfehler
- Wird aber auch oft vom Compiler erkannt

```
1 type Small_Integer is Integer range 0 .. 31;
2
3 function Sum(First : Small_Integer; Second : Small_Integer)
4   return Small_Integer is
5   Ret : Small_Integer := First + Second;
6   begin
7     return Ret;
8   end;
9
10 Value : Small_Integer := Sum(20, 12); -- CONSTRAINT_ERROR
```



Subtypprädikate

- Schränken Typen weiter ein
- Ggf. auch mit Hilfe komplexer Ausdrücke
- Fehlermeldung spätestens zur Laufzeit
- Oft auch schon zur Übersetzungszeit

```
1 type Day is (Monday,
2             Tuesday,
3             Wednesday,
4             Thursday,
5             Friday,
6             Saturday,
7             Sunday);
8 type Weekend_Day is new Day
9   with Static_Predicate => Weekend_Day in
10                                Saturday | Sunday;
11 M_Weekend_Day : Weekend_Day := Monday; -- Fehler, Compiler warnt
12
13 subtype Even is Integer
14   with Dynamic_Predicate => Even mod 2 = 0;
15 Number : Even := 1; -- Laufzeitfehler
```



Typinvarianten

- Überprüfung nicht bei Zugriff auf einzelne Record-Teile
- Nur bei Übergabe/Rückgabe an/aus
 - Funktion
 - Prozedur

```
1 type Day is new String (1 .. 10);
2 type Message is record
3   Sent      : Day;
4   Received  : Day;
5 end record with
6   Dynamic_Predicate => Message.Sent <= Message.Received;
7
8 M : Message := (Received => "1776-07-04", Sent => "1783-09-03");
9 -- Fehler
```



■ Verletzte Verträge werden spätestens zur Laufzeit erkannt

```
1 function Maximum(A : Integer; B : Integer) return Integer
2   with Pre => True,
3        Post => Maximum'Result >= A and Maximum'Result >= B;
4
5 function Maximum(A : Integer; B : Integer) return Integer is
6   Ret : Integer := Integer'First;
7 begin
8   if A > B then
9     Ret := A;
10  else
11    Ret := B;
12  end if;
13  return Ret;
14 end Maximum;
```



```
1 type Integer_Array is array (Positive range <>) of Integer;
2
3 function Maximum(A : Integer_Array) return Integer
4   with Pre => True,
5        Post => (for all M in A'Range
6                 => A(M) <= Maximum'Result);
7
8 function Maximum(A : Integer_Array) return Integer is
9   Ret : Integer := Integer'First;
10 begin
11   for I in A'Range loop
12     assert(for all J in A'First .. I - 1
13            => Ret >= A(J));
14
15     if Ret > A(I) then
16       Ret := A(I);
17     end if;
18
19     assert(for all J in A'First .. I - 1
20            => Ret >= A(J));
21
22   end loop;
23
24   return Ret;
25 end Maximum;
```

