Warum (kurze) Arrays in Fortran (und C/C++) nicht immer die richtige Wahl ist

Thomas Huxhorn

24. Januar 2016

Inhaltsverzeichnis

Zusammenfassung

In diesem kleinen Artikel lege ich ein paar Gedanken nieder über einen Fehler mit (kurzen) Arrays, der systematisch immer wieder Auftritt. Die Wahl der Programmiersprache ist unabhängig für diese Art von Fehler.

1 Ein Fehler

Der Fehler selbst lässt sich in einem Satz beschreiben: Einer Funktion wurde nur ein Wert übergeben, obwohl sie ein Array von zwei Werten erwartet. Das klingt erstmal recht banal, hat es aber in sich. Hier der entsprechende Beispiel Code in Fortran:

```
subroutine func(data)
  integer, intent(in) :: data(2)
  write(*,*) data(1), data(2)
end subroutine
```

```
program test
  integer :: data(2)
  data(1) = 1
  data(2) = 2
  call func(data(1))
end program
```

Das Programm compiliert ohne Warnings, ohne Laufzeitfehlermeldungen und keinerlei Überprüfungen die der Compiler bereit stellt schlagen Alarm. Sogar die Ausgabe des Programms ist richtig. Benutzt wurde der neuste GNU Fortran Compiler in der Version 5.2.1

```
$ gfortran -Wall -Wextra -fcheck=all
-fsanitize=address,undefined wkfanidrwi1.f95
$ ./a.out
1           2
```

2 Ein unsichtbarer Fehler

Auch der memory error detector valgrind findet keinerlei Fehler. Der Grund ist ganz einfach: Aus technischer Sicht existiert in diesem Programm auch kein Fehler! Das ist auf den ersten Blick etwas verwirrend. Der Programmierer wollte, dass die Subroutine func() nur die erste Zahl im Array data verarbeitet. Die Subroutine hingegen, erwartet nicht nur zwei Zahlen, sie nimmt sich auch zwei Zahlen! Selbst, wenn man ihr nur eine gibt. Wir haben hier also gleichzeitig zwei Arten von Fehler. Der Programmierer hat eine Vorstellung, was passieren soll. Seine Kenntnis über die Subroutine func ist aber falsch, so dass er zu wenig Daten übergibt. Das ist der erste Fehler. Der zweite Fehler ist, dass der Programmierer die Subroutine so erstellt hat, dass sie immer genau so viele Daten verarbeitet, wie sie braucht. Egal ob sie auch genügend Daten bekommt. Die beiden Fehler heben sich gegenseitig auf, und es kommt auch das richtige Ergebnis raus. Kein Programm kann den Fehler finden.

3 Ein versteckter Fehler

Heutzutage sind Programme riesig. Sie bestehen aus einer Millionen Zeilen Code, sind über Jahrzehnte gewachsen und kein Mensch blickt mehr in allen Details durch. Diese Art Fehler zu erkennen ist also menschlich nicht möglich. Ein Computer Programm könnte es aber. Der Compiler sieht beim Compilieren das gesamte Programm, den kompletten Quellcode. Man muss den Compiler aber auch seine Arbeit tun lassen, und nicht selbst die Arraygröße bestimmen.

4 Lösungversuch

Die erste Idee ist also, das die Subroutine nicht ein Array fester Größe, sondern eins variabler Größe erwartet. So kann zur Laufzeit erkannt werden, ob zu viele oder zu wenige Daten übergeben wurden. Hier eine Beispiel Implementierung.

```
module testmodule
contains
subroutine func(data)
integer, intent(in) :: data(:)
write(*,*) data(1), data(2)
end subroutine
end module

program test
use testmodule
integer :: data(2)
data(1) = 1
data(2) = 2
call func(data(1))
! call func(data(1:1))
end program
```

Sie Subroutine muss dazu in ein Modul gepackt werden, sonst können keine assumed-shape Arrays benutzt werden.

```
$ gfortran -Wall -Wextra -fcheck=all wkfanidrwi1.f95
wkfanidrwi2.f95:14:12:

call func(data(1))
1
Error: Rank mismatch in argument 'data' at (1)
(rank-1 and scalar)
```

Schon allein das Compilieren führt hier zu einem Fehler. Aber nur, weil ein Array erwartet, aber ein Skalar übergeben wurde. Das kann man ganz leicht aus tricksen, in dem man ein Array der Länge 1 übergibt. Dieser Fall wird dann, wie erwartet, zur Laufzeit abgefangen.

```
$ ./a.out
At line 5 of file wkfanidrwi2.f95
Fortran runtime error: Index '2' of dimension 1 of array
'data' above upper bound of 1
```

Der Fehler wird jetzt zur Laufzeit erkannt. Das ist schon einmal ein riesiger Fortschritt, als wenn der Fehler nie erkannt wird. Allerdings muss nun bei jedem Array Zugriff die Länge des Arrays überprüft werden. Im Durchschnitt verdoppelt das die Laufzeit des Programms. Und der Fehler wird auch nur gemeldet,

wenn entsprechende Checks im Compiler eingeschaltet wurden.
Können wir das nicht besser machen? Immerhin WISSEN wir ja, wie viele Daten die Subroutine brauch. Das muss sich ausnutzen lassen.

5 Lösung allgemein

Die allgemeine Lösung für diese Art von Fehler besteht darin, einfach keine Arrays für so wenige Daten zu nutzen. Um dennoch nicht jede einzelne Zahl als Parameter zu übergeben, und damit den Code unnötig groß zu machen, werden die Daten in einem neuen Datentyp gruppiert.
Da der Datentyp zwangsläufig einen Namen braucht, können wir den Variablen gleichzeitig eine Bedeutung geben. So ist ein weiterer Fehler ausgeschlossen, dass man z.B. aus versehen data(1) statt data(2) benutzt. Hier die Beispiel Implementierung:

```
module testmodule
type Uhrzeit_t
integer :: minute, stunde
end type
contains
subroutine func(uhrzeit)
type(Uhrzeit_t), intent(in) :: uhrzeit
write(*,*) uhrzeit%stunde, uhrzeit%minute
end subroutine
end module

program test
use testmodule
type(Uhrzeit_t) :: uhrzeit
uhrzeit%minute = 1
uhrzeit%stunde = 2
call func(uhrzeit)
end program
```

Der ursprüngliche Fehler ist nun nicht mehr möglich. Unnötig Laufzeiteinverschlechterungen sind auch nicht mehr vorhanden. Der Compiler kann durch Typenüberprüfungen zur Compilezeit feststellen, ob die Subroutine immer den richtigen Datentype erhält.

6 Ausblick

So weit so gut. Mit der Implementierung dem Uhrzeit Modul haben wir aber viele weitere Fehlerquellen erschaffen. In der Praxis reicht es nicht einfach nur

Stunde und Minute auszugeben. Datum und Uhrzeit müssen verglichen, umgerechnet, verrechnet und erstellt werden. Für all das existieren schon fertige Module, teilweise standardisierte und alle getestet.

7 Lösung Qt

Qt bietet hierfür die Klasse QTime an. Hier ein Beispiel wie ein Zeitobjekt erstellt, einer Funktion übergeben und ausgegeben wird.

```
void func(QTime time) {
        qDebug() << time.toString();
}

QTime time(1,2);
func(time);
```

QTime selbst ist eine Klasse welche nicht nur Stunden, Minuten, Sekunden und Millisekunden speichern kann. Sondern auch die Textausgabe im jeweiligen Landesformat übernimmt und mit der Klasse QDateTime werden auch Winter und Sommerzeit beachtet.

8 Technische Details des Fehlers

Wie am Anfang erwähnt ist diese Fehlerart unabhängig der gewählten Programmiersprache. Auch in C können Arrays als Funktions Parameter eine feste Länge zugewiesen werden. Und da C++ rückwärts kompatibel zu C ist, ist der Fehler auch in C++ machbar. Es hängt nicht von der Sprache ab, sondern die Art wie man programmiert.
In der Subroutine wurde ein Array fester Länge angegeben. Damit existiert das Array für den Compiler! und er würde so nie einen Fehler finden. Warum die ganzen Memory Check Programm auch keinen Fehler finden, ist recht einfach. Es existiert im Programm ja auch ein Array richtiger Länge. Es wurde nur ein statt zwei Elemente übergeben. Aber in dem Moment wo in der Subroutine auf das zweite Element zugegriffen wird, wird gültiger Speicher ausgelesen.