

Systemnahe Programmierung

Vorlesung

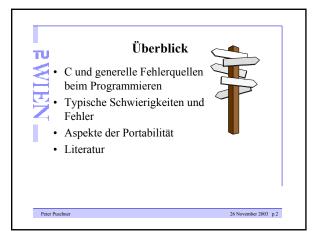
C – Tücken und Programmierkonventionen

Peter Puschner

Institut für Technische Informatik



26/11/03



Gründe für die Verwendung von C

- · Compiler Verfügbarkeit für viele Prozessoren
 - z.T. einzig verfügbare Hochsprache
 - Portierbarkeit von Code
- Schneller Code, low-level Programmierung
- Kleinerer und schnellerer Code als bei anderen Sprachen
- Viele Code-Generatoren erzeugen C Code (z.B., Matlab/Simulink RTW, dSPACE Targetlink)

Peter Puschner

26 November 2003 p.3



C Standard

- ISO/IEC 9899, 1999
- Definiert Syntax und Semantik der Sprache
- · Definition ist nicht vollständig
 - Anhang: Portability, Unspecified behaviour
 - Anhang: Portability, Undefined behaviour

Peter Puschner

26 November 2003 p.4



Fehlerquellen beim Programmieren

- Programmierer macht Fehler
 - Stil und Ausdrucksmöglichkeiten der Sprache
 - Robustheit der Sprache gegen Fehler
 - Wie schwer ist es, durch einen Fehler aus einem gültigen Konstrukt ein anderes gültiges Konstrukt zu erzeugen?
 - 2. Fehlererkennung durch die Sprache

Peter Puschne

26 November 2003 p.5

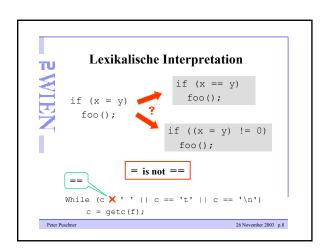
Fehlerquellen beim Programmieren (2)

- · Missverständnisse durch Sprache
 - z.B. Operator Precedence
- Compiler macht nicht das, was der Programmierer erwartet
 - undefiniertes Sprachverhalten
- · Fehler im Compiler
 - Falsche Interpretation des Standards
 - Implementierungsfehler
 - Bewusste Abweichungen vom Standard

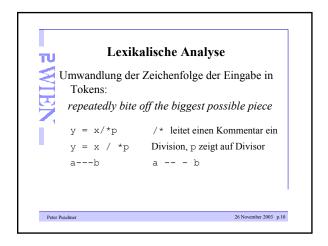
Peter Puschner

26 November 2003 p.6









```
Syntaktische Eigenheiten

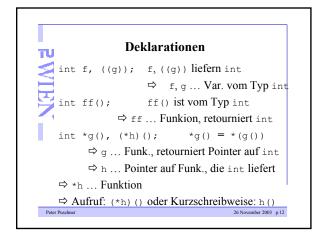
• Funktionsdeklarationen und Aufrufe

(* (void(*)())0)(); ??

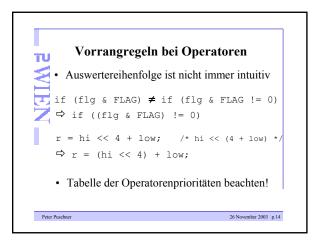
Declare it the way you use it

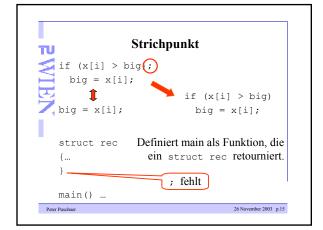
• Deklaration = Typ + Deklaratoren

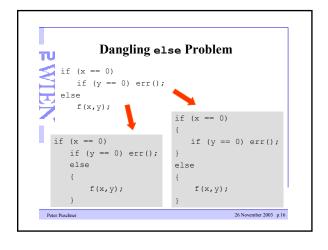
• Deklarator: Ausdruck, der angegebenen Typ retourniert
```



Deklaration und Typkonversion (Cast) Cast: wie Deklaration, aber ohne Variablenname, ohne Strichpunkt und in runden Klammern Deklaration: int (*h)(); Cast: (int (*)()) Aufgabe: Aufruf einer Funktion, deren Adresse an der Speicheradresse 0 gespeichert ist. 1. Versuch: (*0)() 0 hat falschen Typ Benötigtes Objekt: void (*fp)(); Lösung mit Cast: (*(void(*)())0)();

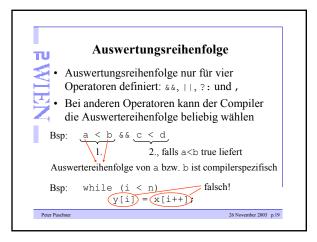






Pointers und Arrays C kennt nur eindimensionale Arrays Arraygröße wird zur Compilezeit fixiert Zwei Arrayoperationen Ermitteln der Größe des Arrays Retournieren der Adresse von Element 0 Alle anderen Operationen werden durch Pointeroperationen realisiert (z.B. Indizierung) (weitere Details: siehe Folien zum C-Block)





Portabilität

• Namen: In ANSI C sind die ersten sechs Zeichen für die Unterscheidung externer Namen signifikant (Keine Unterscheidung von Groß- und Kleinschreibung)

Integer-Typen: short, plain, long

TU

- Größen: short ≤ plain ≤ long ANSI C: short, plain ≥ 16 Bit, long ≥ 32 Bit
- Plain Integer ist groß genug für Array-Indizierung
- Character: Größe entspricht "natürlicher", adressierbarer Einheit (8, 9, 16 Bit)

Peter Puschner 26 November 2003 p.20

TU

Portabilität (2)

- Characters: die meisten Compiler realisieren Characters als 8-bit Integers
- Konvertierung char → int: es ist undefiniert, ob ein Character als signed oder unsigned betrachtet wird.
- Shift-Operationen
 - Wie werden Bits beim Rechts-Shift aufgefüllt? → signed oder unsigned definieren.
 - Max. Shift-Count: n-1 Bits bei Typ der Größe n

Peter Puschner

26 November 2003 p.21

Portabilität (3)

Ganzzahlige Division q = a / b;

r = a % b;

1. q*b + r == a2. a negativ → q negativ, |q| unverändert 3. b>0 → r>=0 und r<b

Eigenschaften 1-3 nicht gleichzeitig erfüllbar

Bsp.: (-3)/2

- (2.): $q==-1 \rightarrow (1.)$: $r==-1 \rightarrow Widerspruch zu (3.)$

- (3.): $r==1 \rightarrow$ (1.): $q==-2 \rightarrow$ Widerspruch zu (2.)

→ Die meisten Compiler geben Eig. 3 auf, einige Eigenschaft 2.

Peter Puschner 26 November 2003 p.22

Programmierrichtlinien in der LU TU

- Checks beim Compilieren
 - Fehlerabfragen im Programm
 - Namenskonventionen
 - Defensive Programmierung (keine komplizierten Ausdrücke, keine globalen Variablen, etc.)
 - Ressourcensparend arbeiten
 - Standardfunktionen verwenden

26 November 2003 p.23

Literatur

- Andrew Koenig, C Traps and Pitfalls, Addison Wesley, 1988
- Les Hatton, Safer C, McGraw-Hill, 1994

Richtlinien für sicherheitskritische Anwendungen

- MISRA, Guidelines for the Use of the C Language in Vehicle Based Software, 1998
- Hecht, et al., Review Guidelines for Software Languages for Use in Nuclear Power Plant Systems, US Nucl. Reg. Comm., 1997 (WWW)

Was haben wir gelernt?

- C ist eine mächtige, weit verbreitete Programmiersprache
 - Für das Schreiben korrekter und sicherer Programme ist eine gute Kenntnis der Sprache erforderlich
 - Typische Probleme wurden vorgestellt
 - Programming Guidelines



Peter Puschner

TU

26 November 2003 p.25