



Prof. Dr. Aris Christidis

Konzepte Systemnaher Programmierung

Prof. Dr. Aris Christidis

SS 2009

Organisatorisches



Prof. Dr. Aris Christidis

- Übungen (SS 09) freiwillig: di. 3. Block; fr. 1. Block
- Programmieren in C mit MS Visual Studio 2005 Vers. 8.0
- Eigener Name als erste Zeile der Programmausgabe (`printf`)
- In allen Quellen: Name & Matr.Nr. im Kopf als Kommentar
- Abgabe: Dateien *.h, *.c, *.exe, *.sln, *.vcproj (*.dsw, *.dsp) – vorzugsw. in Verzeichnis-Struktur. Bitte nicht zumüllen!
- Per Email gezippt (z.B. KSPuebg3.zip) bis zum Abgabetermin einsenden (auch an Tutor! **3** Wochen – s.a. www)
- Keine (positiven) Benachrichtigungen vorgesehen!
- Alle Übungen zu 100% gelöst: 10% der Punkte für „1,0“

Literatur



Prof. Dr. Aris Christidis

- Übungsblätter, frühere Klausuren (inkl. Lösungshilfe) Materialien, Programme, Vorlesungsfolien (KEIN Skript):
<http://homepages.fh-giessen.de/christ/>
- MSDN Library for Visual Studio (Microsoft Developer Network)
- Regionales Rechenzentrum für Niedersachsen (RRZN) / Softwareberatung Herdt:
„Die Programmiersprache C. Ein Nachschlagewerk“, RRZN 1995
(<http://www.rrzn.uni-hannover.de/Dokumentation/Handbuecher/index.html>)

Bezug: Hochschulrechenzentrum der Justus-Liebig-Universität Gießen
Heinrich-Buff-Ring 44, 35392 Gießen, Tel. 0641/99-13017 („Kaufladen“)
Wegbeschreibung: <http://www.uni-giessen.de/hrz/organisation/weg.html>



Literatur

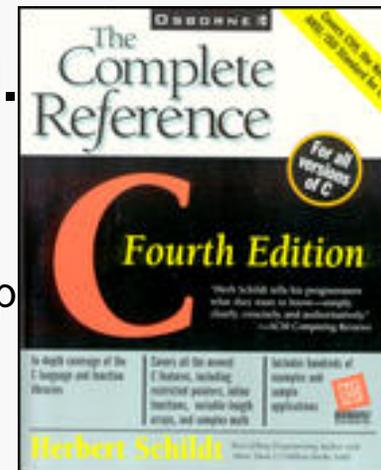


Prof. Dr. Aris Christidis

- B.W. Kernighan, D.M. Ritchie:
„Programmieren in C“
(2.Ausgabe, ANSI-C), Carl Hanser 1990

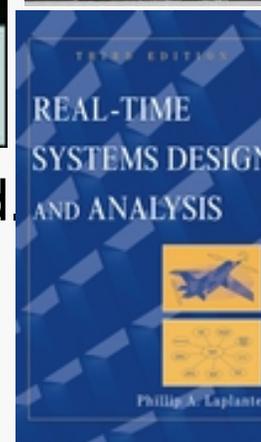


- H.Schildt:
„C-The Complete Reference“, 4th Ed.
McGraw-Hill 2000, \$39,99



- **„C Ent-Packt“**, mitp, 2001, €39,95
www.mitp.de/imperia/md/content/vmi/0732/0732_listings.zip

- Ph. A. Laplante:
"Real-Time Systems – Design and Analysis" 1st Ed.
IEEE Press 2004, \$105,00



Literatur



Prof. Dr. Aris Christidis

- J.Nehmer, P.Sturm:
„Systemsoftware – Grundlagen moderner Betriebssysteme“
dpunkt-Verlag 2001
- Andrew S. Tanenbaum:
„Modern Operating Systems“
(2nd Edition) Prentice Hall 2001 – bzw.:
„Moderne Betriebssysteme“
(2. Auflage) Prentice Hall 2002, €49,95
- Ch. Petzold:
„Windows Programmierung“
(5.Auflage) Microsoft Press 2000, €59,90



Allgemeines



Prof. Dr. Aris Christidis

Vorlesung (vorerst) stark angelehnt an ehem. SysProg 2:

Aspekte* der Struktur u. Funktionsweise von Sw-Plattformen und -Umgebungen

(*) Beispielhafte Sw-Techniken

- Systeme und ihre Programmierung
- C-Programmierung
- Timer, Callbacks, Globale Variablen
- Prozeß-Interaktion u. –Kommunikation
- Blockierungsarten, Vermeidungsstrategien
- Ereignisse und Fenstersysteme
- Thread-Programmierung



Hier verwendete Begriffe:

- **Sw-Plattform:**
Die ‚tiefste‘ (Hw-nächste) genutzte Sw-Schicht
- i.d.R. das Betriebssystem
- **Sw-Umgebung:**
Satz von Programmen, die für eine bestimmte Nutzung die Kommunikation mit der Plattform übernehmen
(vgl. Sw-Paket, -Bibliothek, -Entwicklungsumgebung, OpenGL, frühe Windows-Versionen)
- Ein **Programm** ist die Planung einer Reihe von Handlungen (einschließlich Teilhandlungen und Details), die auf ein einheitliches Ziel hinsteuern.



- Definition: Als **System** bezeichnen wir
 - Eine Menge von **Komponenten**
(Gegenständen, Individuen, Größen, Prozessen, Ideen),
 - die untereinander in einer **kausalen Wechselwirkung** stehen
 - und von ihrer **Umgebung** entweder als abgeschlossen oder als in einer wohldefinierten **Beziehung** stehend betrachtet werden können
 - sowie die Gesamtheit der **unter ihnen** herrschenden **Beziehungen**.

Systeme



Prof. Dr. Aris Christidis

- Im Sinne dieser Definition kann jedes System in beliebig viele **Teil- oder Subsysteme** zerlegt werden, sofern dies sinnvoll erscheint.
- Die Existenz von Systemen ist meist mit der Erfüllung eines **Zweckes** verbunden (vgl. technische, politische, soziale, Regal-, Gleichungs- oder Planetensysteme).
- Der Begriff “System” ist ein **Idealtypus**:
Komponenten sind selten abgeschlossen, nicht immer unterscheidbar, Beziehungen kaum einmal “wohldefiniert”
z.B.: Verkehrssysteme, politische Systeme –
Ggs.: philosophische, mathematische (Gleichg.-) Systeme
- **Software**-Systeme lassen sich in sehr guter Näherung durch **Idealtypen** beschreiben (trotz Digitalisierungsfehler, Fremdeinflüssen durch Betriebssystem u.ä.)

Systeme



Prof. Dr. Aris Christidis

Drei Software-Schichten des Sw-Gesamtsystems:

- **Anwendungsprogramme:** } Sw / Mensch
- **Systemprogramme:** } Sw / Sw
Betriebssystem
Kommando-Interpreter,
Editoren, Compiler
- **Hardware-Ansteuerung:** } Hw / Sw
Physikalische Geräte
Mikroprogrammierung
Maschinensprache



Meistverwendete Sprache in der systemnahen Programmierung: C

(Memo.exe)

Die Programmiersprache C: eine Übersicht



Prof. Dr. Aris Christidis

Zur Erinnerung:

- Ab ca. 1950 lösen Assemblersprachen die Maschinensprache ab; sie sind auch prozessorabhängig, strukturäquivalent zu Binärcode, verwenden aber mnemonische Bezeichner.
- 1954-...FORTRAN: rechnerunabhängige Sprache: Syntax (Schreibweise) / Semantik (Bedeutung) anwendungsabh.; prozedural (*procedure-oriented*): Faßt Maschinenbefehle zu logischen Anweisungen; Trennung Daten ↔ Programm
- 1970-: allmählich Einzug v. C: prozedural, OOP-Elemente

C-Syntax vergleichbar zu Java (1995-); jedoch kennt C

- keine Klassen (sondern Strukturen)
- keine Referenzen (sondern Zeiger / Adressen)
- keine autom. Speicherbereinigung (obliegt Programmierer)

Die Programmiersprache C: eine Übersicht



Prof. Dr. Aris Christidis

- C-Programm: Aufruf-Folge von Funktionen, die bel. viele Daten aufnehmen und max. je einen Wert zurückgeben (für keine Daten oder Rückgabewert: `void`)
- Funktionen: Auswahl aus (wenigen Dutzend) C-Anweisgn und (Hundertern von) Bibliotheks- u. Anwenderfunktionen
- Genau eine anwenderdefinierte Fkt. muß `main()` heißen: Name=Startadresse für Ausführung; weltbekanntes Bsp.:

```
#include <conio.h> //wg._getch()
#include <stdio.h> //wg.printf()
int main (void)
{ printf("Hello, world!\n\r");
  _getch();
  return 0;
}
```



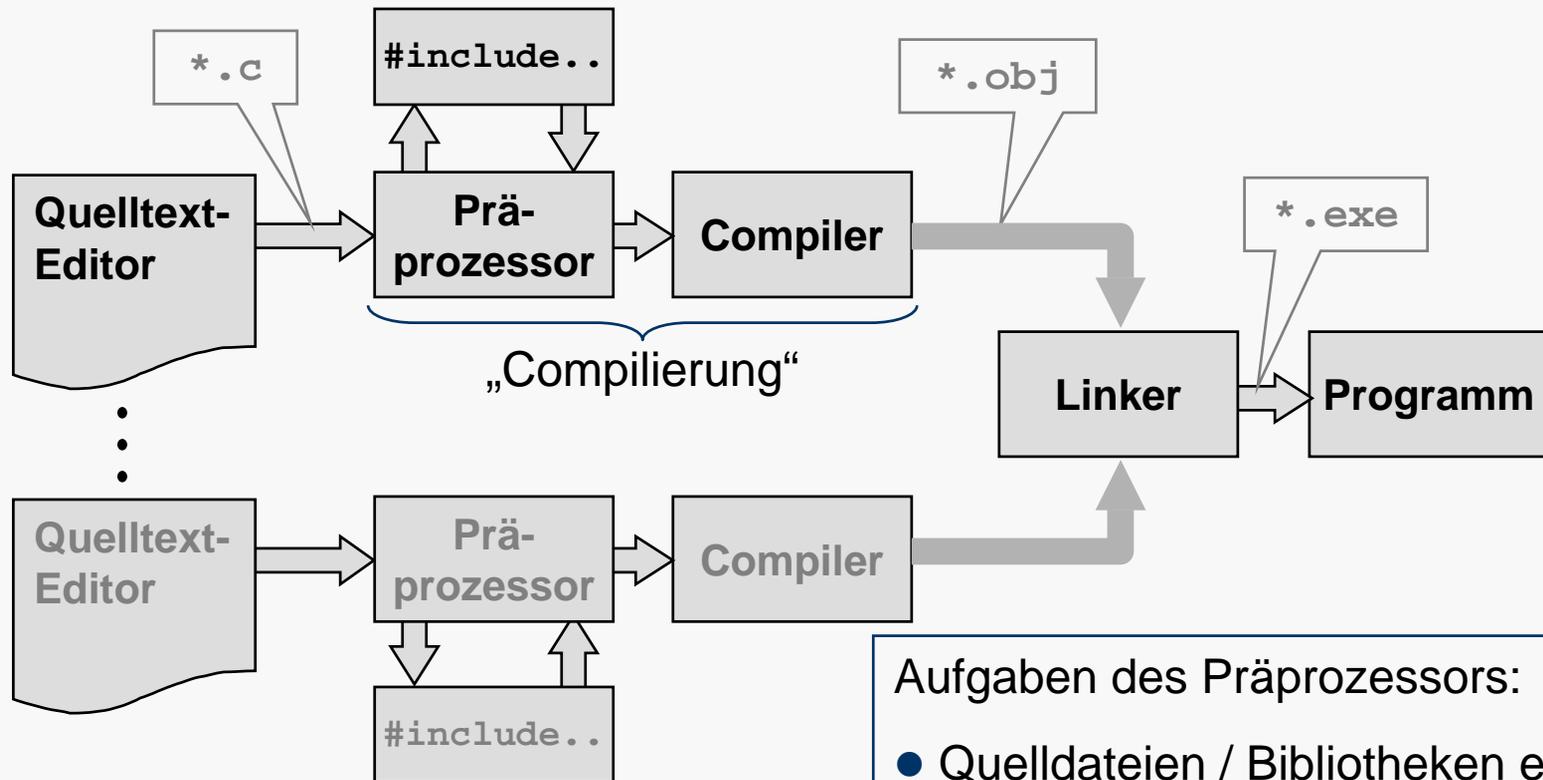
(HelloWld)

Die Programmiersprache C: eine Übersicht



Prof. Dr. Aris Christidis

Erstellung von Maschinencode aus C-Quelle(n):



- Aufgaben des Präprozessors:
- Quelldateien / Bibliotheken einfügen
 - Bedingte Compilierung
 - Makro(-Befehl-)Definition
 - Compileroptionen

„Direktiven“ (Präprozessor-Anweisungen) werden mit Raute (#) eingeleitet

Die Programmiersprache C: eine Übersicht



Prof. Dr. Aris Christidis

Wichtige Präprozessor-Einsätze:

Bedingte Compilierung

- ... zur Vermeidung rekursiver/mehrfacher Definitionen u.ä.:

```
#ifndef HEADER_H
#define HEADER_H /*... #include ...*/
#endif//HEADER_H
```

- ... zur Unterscheidung von Sw-Versionen:

```
#ifdef WIN32
    sleep(1000); //msec
#else //WIN32
    sleep(1);    // sec
#endif//WIN32
```

Konstanten- / Makro-Definitionen (textliche Austauschung):

```
#define ESC 27
#define MAX(x,y) (( (x) > (y) ) ? (x) : (y))
```

Die Programmiersprache C: eine Übersicht



Prof. Dr. Aris Christidis

```
#include <conio.h> bzw. #include <stdio.h>
```

Eingefügte Header-Dateien (*.h) machen verwendete Standard-Funktionen u. -Bibliotheken (<*.h>) oder eigene Funktionen/ Bibliotheken ("*.h") dem Compiler bekannt.

```
int main (void)
```

Auch zulässig: `int main (int argc, char *argv[])` für Aufruf aus der Kommandozeile mit `argc` Argumenten (Aktualpar.), übergeben als Feld v. Zeichenfolgen `argv[]`

```
printf ("Hello, world!\n");
```

* : Inhalt von...

- `int printf (const char* format [,var1, ...]);`

Konsole-Ausgabe [übergebener Variablen `var1,...` in] einer Zeichenkette mit Text und Format-Anweisungen – analog:

- `int scanf (const char *format [,&var1, ...]);`

Rückgabewert von `printf()` / `scanf()`:

& : Adresse von...

Anzahl jeweils erfolgreich aus-/eingegebener Datenobjekte

Die Programmiersprache C: eine Übersicht



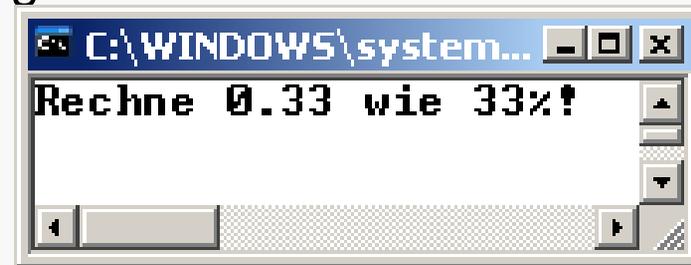
Prof. Dr. Aris Christidis

Interessante Formate für `printf()` / `scanf()`:

- `%c` Character (einzelnes Schriftzeichen)
- `%d` ganze Zahl (Dezimaldarstellung: `%3d`: 3stellig)
- `%f` Gleitkommazahl; (`%10.6f`: 10 Zeichen, 6 Nachkommastellen)
- `%g` Gleitkommazahl: kürzeste Form, volle Genauigkeit
- `%p` Adresse (im rechner eigenen Format)
- `%s` String (Zeichenkette bis zum abschließenden `'\0'`)
- `%x` ganze Zahl (Hexadezimaldarstellung)
- `%lf`, `%ld` long float (`double`, 8 Byte), long decimal

"\...":
Escape-
Sequenzen

- `\b` backspace
- `\n` new line
- `\r` carriage return (CR: Wagenrücklauf)
- `\t` Tabulator
- `\0`, `\'`, `\"`, `\\`, `%%`: das jeweils zweite Zeichen
- `\(3 Oktalziffern)`: Bitmuster: `"7"` für `'7'`, `"\007"` für Klingel (`"\a"`)



Bsp.: `float part=1.f/3; //keine int- o.double-Div.!`
`printf("Rechne %4.2f wie%3d%%!\007\n\r", part,`
`(int)(part*100));` (HelloWld)

„Cast-Operator“

Die Programmiersprache C: eine Übersicht



Prof. Dr. Aris Christidis

Flexible Formatierung von `printf()` / `scanf()` durch Sternchen ('*') u. Variablen (bzw. Konstanten) statt Zahlen:

```
/*... #include ...*/
#define MESSAGE "Hallo - zum "
int main (void)
{ int j=1; /*...*/ j++;
  printf("%s%d%s", MESSAGE,
        (int)log10(j)+1, j, ".!\n\r"); }      (HelloWld)
```



- `int _getch (void);` /*auch: `getch()`; puffert!*/
Einlesen v. Zeichen ohne Ausgabe (Echo); kein Standard, aber nahezu überall implementiert; gibt ASCII-Wert zurück
C-Standard: Funktion, die Zeichen von Tastatur liest und auf Konsole ausgibt; erwartet abschließendes <Enter> (=ASCII(LF)=10 \leftrightarrow `_getch`: <Enter>=ASCII(CR)=13)
`int getchar (void);` /*Eingabe-Pufferung!*/

Die Programmiersprache C: eine Übersicht

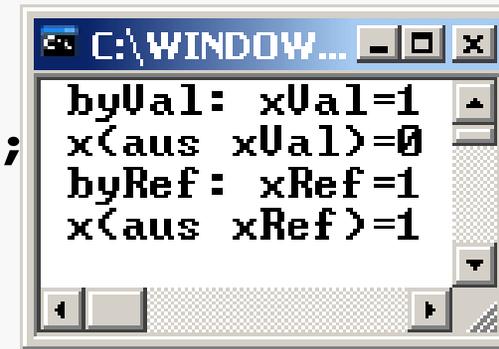
Prof. Dr. Aris Christidis

- Mächtig, aber gefährlich: Zeiger (Speicheradressen)
2 Zeigeroperatoren: & [Adresse von...] und * [Inhalt von...]
Zeiger ermöglichen (u.a.) die Änderung der Argumente in Funktionsaufrufen (Call by value vs. by reference) – Bsp.:

```
void incByVal (int xVal)
{ xVal++;
  printf(" byVal: xVal=%d\n\r", xVal);
  return; }
```

```
void incByRef (int *xRef)
{ (*xRef)++; //Klammern!
  printf(" byRef: xRef=%d\n\r", *xRef);
  return; }
```

```
int main()
{ int x=0;
  incByVal( x);printf(" x(aus xVal)=%d\n\r", x);
  incByRef(&x);printf(" x(aus xRef)=%d\n\r", x);
  _getch(); return (0); }
```



```
C:\WINDOW...
byVal: xVal=1
x(aus xVal)=0
byRef: xRef=1
x(aus xRef)=1
```

(ValRef)

```
#include <conio.h>
#include <stdio.h>
```

„zeigt auf“ =
„enthält
Adresse von“

Die Programmiersprache C: eine Übersicht



Prof. Dr. Aris Christidis

- Zeiger-Variablen: Speicherzellen, die Speicheradressen von Daten-Objekten (Variablen, Funktionen) aufnehmen. Typunterscheidung berücksichtigt Variablengröße (i.Byte); `void *` als generischer (Universal-) Zeiger.

Interessant: Initialisierung mit `NULL` (reservierte Adresse).

- Zulässige Zeigeroperationen: Wertzuweisung /-vergleich, Integer-Addition/-Subtraktion und Zeiger-Subtraktion
- Felder: hintereinander gespeicherte (indizierte) Variablen
⇒ Allgemein: `Wert[Index] == *(Wert+Index)`

Indizierung oft langsamer als Zeigerarithmetik (syst.abh.)!

Feldname = Feld-Anfangsadresse = Adresse v.Element[0]

- Nutzung bei Speicherplatz-(Heap-)Reservierung für (auf null gesetzte) Datenobjekte bekannter Größe – z.B.:

```
int *p2i; p2i=(int*)calloc(1,sizeof(int));  
if (p2i) p2i[0]=1; free (p2i);
```

liefert void*

Übung:



Prof. Dr. Aris Christidis

- Verbessern Sie die vorhandene Implementierung eines einfachen FILO-Speichers (Stack) so, daß er seine gesamte Speicherkapazität nutzen kann.
- Weitere Hinweise und Hilfen: s. Übungsblatt.

```
C:\WINDOWS\system32\cmd.exe
Stack speichert bis zu 50 Zahlen (FILO):
Geben Sie einen Wert an: 1
Geben Sie einen Wert an: 2
Geben Sie einen Wert an: 3
Geben Sie einen Wert an: 0
Der oberste Wert ist 3
Geben Sie einen Wert an: 0
Der oberste Wert ist 2
Geben Sie einen Wert an: 0
Der oberste Wert ist 1
Geben Sie einen Wert an: -1
Drücken Sie eine beliebige Taste . . .
```

(Stack.exe)