Konzepte
Systemnaher Programmierung

Prof. Dr. Aris Christidis

WS 2012 / 13

Organisatorisches: Übungen

- Übungen freiwillig: fr. 3.-5. Block
- Programmieren in C mit MS Visual Studio (2005 - 2010)
- Lösung aller (5) Übungen ⇨ 10% der Punkte für „1,0“
- Auslassung einer Übung ⇨ 5% der Punkte für „1,0“
- Eigene(n) Namen & Matr.Nr. in der Programmausgabe (**`printf`**) und in allen Quellen als Kopf-Kommentar
- Vorführung, dann Abgabe: Dateien *.h,*.c,*.exe, *.vcxproj, *.vcxproj.filters (bis VS 2008: *.sln, *.vcproj) – vorzugsw. in Verzeichnis-Struktur. Bitte nicht zumüllen!
- Per Email gezippt (z.B. CGB-Uebg3.zip) bis zum Abgabe-Termin einsenden (auch an Tutor! 5 Wochen – s.a. www)
- Keine (positiven) Benachrichtigungen vorgesehen!

Literatur

- Übungsblätter, frühere Klausuren (inkl. Lösungshilfe) Materialien, Programme, Vorlesungsfolien (KEIN Skript):
  http://homepages.thm.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/buecher.html?&no_cache=1)

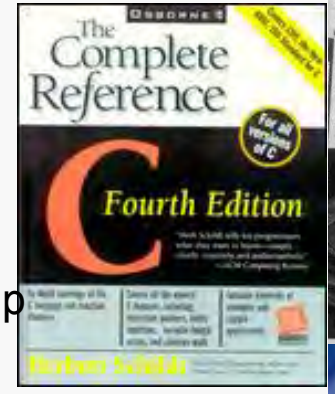
Bezug: Hochschulrechenzentrum der Justus-Liebig-Universität Gießen
Heinrich-Buff-Ring 44, 35392 Gießen, Tel. 0641/99-13017 („Kaufladen“)
Wegbeschreibung: http://www.uni-giessen.de/hrz/organisation/weg.html

Literatur

- 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

- 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

Vorlesung (zunächst) an ehem. SysProg 2 angelehnt:

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

Allgemeines

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.

Systeme

- 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

- 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

Drei Software-Schichten des Sw-Gesamtsystems:

- **Anwendung**sprogramme: } Sw / Mensch
- **System**programme:
  Betriebssystem
  Kommando-Interpreter,
  Editoren, Compiler
  } Sw / Sw
- **Hardware**-Ansteuerung:
  Physikalische Geräte
  Mikroprogrammierung
  Maschinensprache
  } Hw / Sw

Meistverwendete Sprache in der systemnahen Programmierung: C

Übersicht: Programmiersprache C

Zur Erinnerung:

- Ab ca. 1950 lösen Assemblersprachen die Maschinenspr. 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)

Übersicht: Programmiersprache C

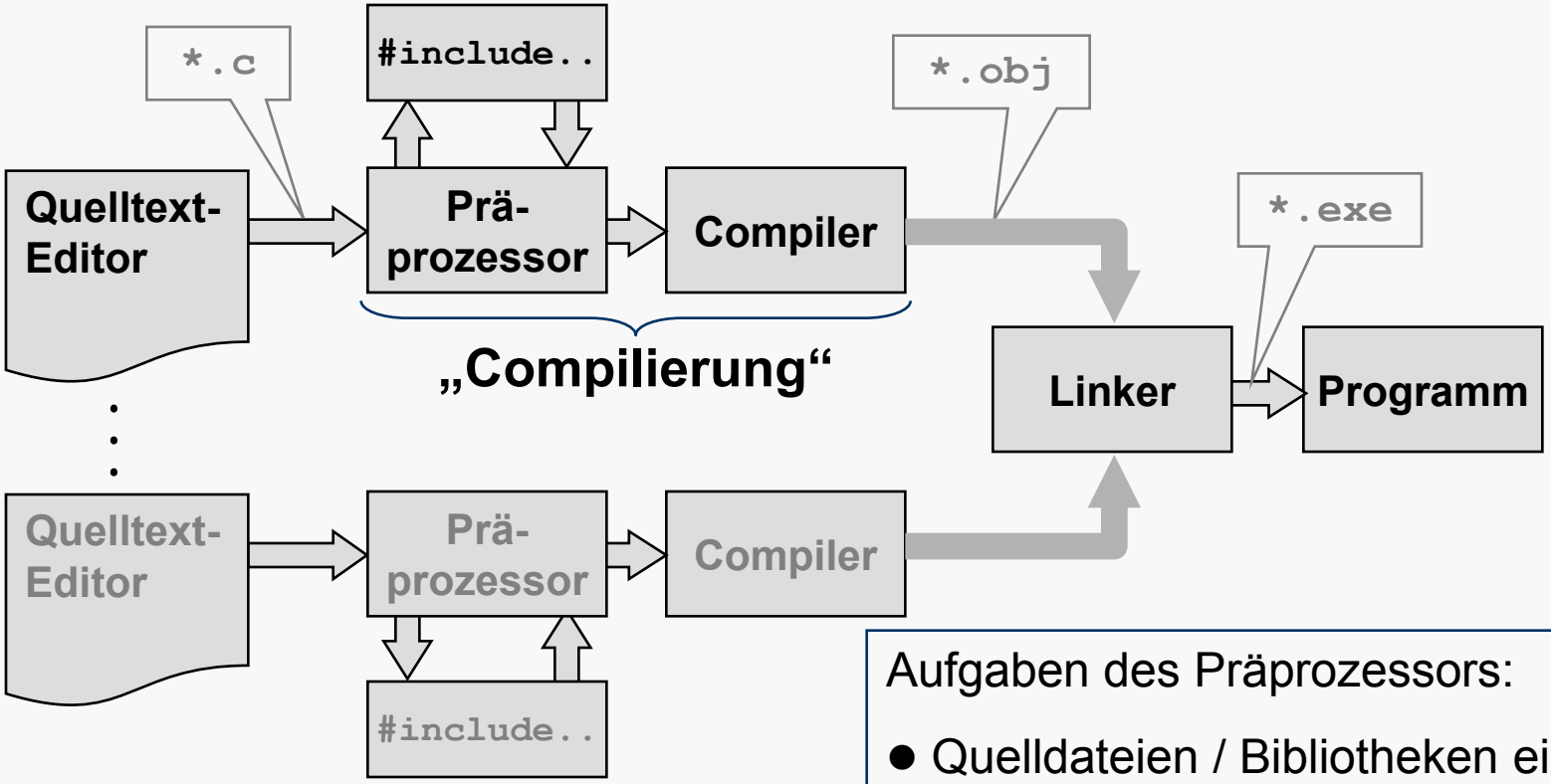
- C-Programm: Aufruf-Folge von Funktionen, die bel. viele Daten aufnehmen und max. je einen Wert zurückgeben (für keine Daten oder Rückgabewert: **`void`**)
- Funktionen: Auswahl aus (wenigen Dutzend) C-Anweisgn und (Hunderten von) Bibliotheks- u. Anwenderfunktionen
- Genau eine anwenderdefinierte Fkt. 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)

Übersicht: Programmiersprache C

Erstellung von Maschinencode aus C-Quelle(n):



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

Aufgaben des Präprozessors:

- Quelldateien / Bibliotheken einfügen
- Bedingte Compilierung
- Makro(-Befehl-)Definition
- Compileroptionen

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))
```

Ohne Leerzeichen!

Übersicht: Programmiersprache C

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

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

`*` : Inhalt von...

```
int main (int argc,char *argv[])
```

`main()` kann aus Kommandozeile aufgerufen werden mit einer Anzahl **`argc`** von „Argumenten“ (Aktualparametern), die als Feld **`argv[]`** von Zeichenfolgen übergeben wird – auch zulässig ohne „Parameter“ (Formalparameter): **`int main(void)`**

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

- **`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()`**: Anzahl jeweils erfolgreich aus- /eingegebener Datenobjekte

`&` : Adresse von...

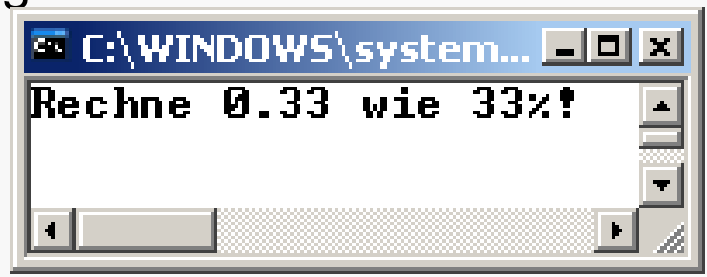
Übersicht: Programmiersprache C

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 rechnereigenen 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));
```

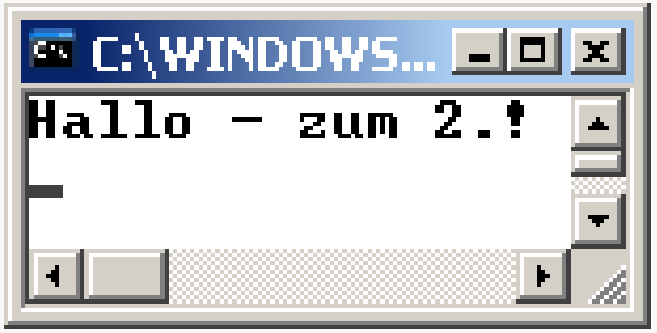
(HelloWld)

„Cast-Operator“

Übersicht: Programmiersprache C

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.!\n\r", MESSAGE,
       (int)log10((double)j)+1, j);
printf("%s%*d%s", MESSAGE,
       (int)log10((double)j)+1, j, ".!\n\r");
```



(HelloWld)

- **`int _getch (void);`** `/*auch: getch();*/`

  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 ⇔ **`_getch`**: <Enter> = ASCII(**`CR`**)=13)

  `int getchar (void);` `/*Eingabe-Pufferung!*/`

Übersicht: Programmiersprache C

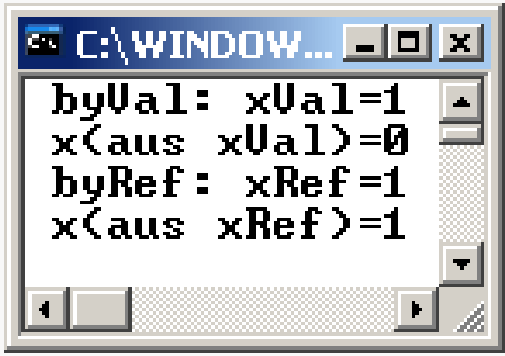
- Mächtig, aber gefährlich: Zeiger (Speicheradressen)

  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); }
```

* : Inhalt von...



(ValRef)

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

Übersicht: Programmiersprache C

„zeigt auf“ = „enthält Adresse von“

- 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.)!

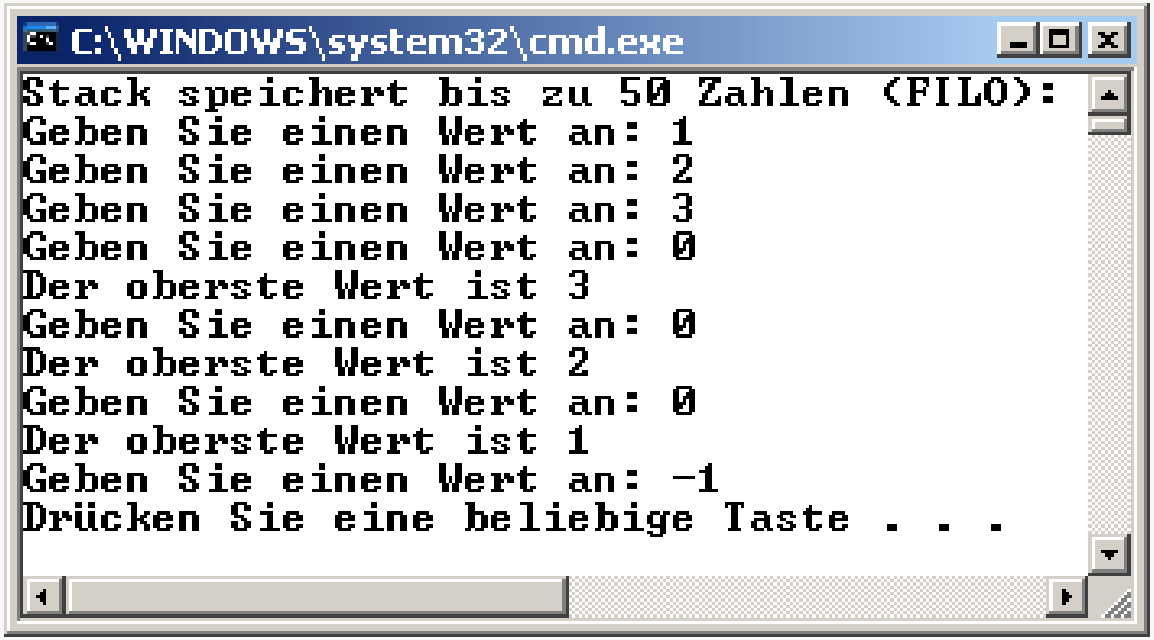
  Feldname = Feld-Anfangsadresse = Adresse v.Element[0]
- Nutzung bei Speicherplatz-(Heap-)Reservierung für (auf null gesetzte) Datenobjekte bekannter Größe – z.B.:

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

liefert **`void*`**

Übung:

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



(Stack.exe)