Slide 1

Einführung in die Programmiersprache C
für Hörer der Betriebssysteme-Vorlesung

URL: `http://os.inf.tu-dresden.de/~hohmuth/ckurs/`

Michael Hohmuth <***hohmuth@inf.tu-dresden.de***>

19. Oktober 2000

Slide 2

1 Überblick über diesen Kurs

3 Doppelstunden:

1. Einführung: Warum C? — Hello World! — Aufruf des Compilers unter Unix — Online-Hilfe unter Unix — Typen — Variablen — Ausdrücke
2. Steueranweisungen — Präprozessor/Makros — Standard-Bibliothek — Module, Bibliotheken, Linker
3. Programmieren mit Zeigern — UNIX-Systemrufe — Objektorientiertes Programmieren in C

Übungsaufgaben: zu 1. und 2. jeweils $\frac{1}{2}$h

Voraussetzungen (wird *nicht* beprochen):

Slide 3

- Algorithmus; Programmierung allgemein
- Einloggen, Ausloggen
- Unix-Editor

2 Einführung

Warum C für Betriebssysteme?

Slide 4

- portabel — Compiler für alle Plattformen verfügbar
- ausgereift — lange Historie als Implementationssprache für Betriebssysteme; Stärken/Schwächen wohlbekannt
- minimalistisch, hardwarenah und effizient — Programmierer verliert nicht den Überblick über Umsetzung in Maschinencode; kein Laufzeitsystem erforderlich
- Syntax unterstützt „faule“ Programmierer (wenig zu tippen)

Schwächen von C

Slide 5

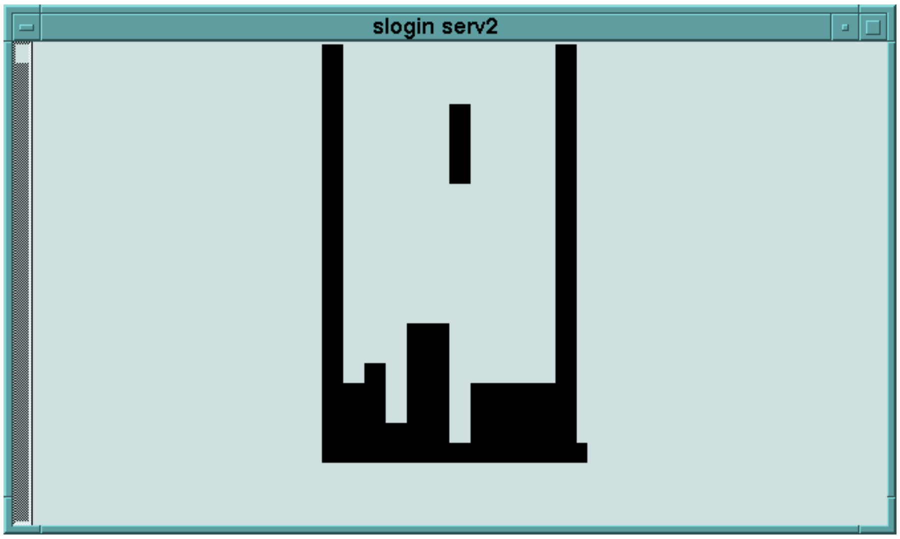
- einige Fallstricke für Anfänger
- erfordert diszipliniertes Programmieren, damit der Compiler Fehler erkennen kann und damit der Code für Fremde lesbar wird
- nur begrenzte Unterstützung für Modularisierung — Programmierer großer Systeme müssen eigene Konventionen erfinden

„C combines the speed of assembler with the readability of assembler.“

Slide 6

```
long h[4];t(){h[3]-=h[3]/3000;setitimer(0,h,0);}c,d,l,v[]={(int)t,0,2},w,s,I,K
=0,i=276,j,k,q[276],Q[276],*n=q,*m,x=17,f[]={7,-13,-12,1,8,-11,-12,-1,9,-1,1,
12,3,-13,-12,-1,12,-1,11,1,15,-1,13,1,18,-1,1,2,0,-12,-1,11,1,-12,1,13,10,-12,
1,12,11,-12,-1,1,2,-12,-1,12,13,-12,12,13,14,-11,-1,1,4,-13,-12,12,16,-11,-12,
12,17,-13,1,-1,5,-12,12,11,6,-12,12,24};u(){for(i=11;++i<264;)if((k=q[i])-Q[i]
){Q[i]=k;if(i-++I||i%12<1)printf("\033[%d;%dH",(I=i)/12,i%12*2+28);printf(
"\033[%dm  "+(K-k?0:5),k);K=k;}Q[263]=c=getchar();}G(b){for(i=4;i--;)if(q[i?b+
n[i]:b])return 0;return 1;}g(b){for(i=4;i--;q[i?x+n[i]:x]=b);}main(C,V,a)char*
*V,*a;{h[3]=1000000/(l=C>1?atoi(V[1]):2);for(a=C>2?V[2]:"als pq";i;i--)*n++=i<
25||i%12<2?7:0;srand(getpid());system("stty cbreak -echo stop u");sigvec(14,v,
0);t();puts("\033[H\033[J");for(n=f+rand()%7*4;;g(7),u(),g(0)){if(c<0){if(G(x+
12))x+=12;else{g(7);++w;for(j=0;j<252;j=12*(j/12+1))for(;q[++j];)if(j%12==10){
for(;j%12;q[j--]=0);u();for(;--j;q[j+12]=q[j]);u();}n=f+rand()%7*4;G(x=17)||(c
=a[5]);}}if(c==*a)G(--x)||++x;if(c==a[1])n=f+4**(m=n),G(x)||(n=m);if(c==a[2])G
(++x)||--x;if(c==a[3])for(;G(x+12);++w)x+=12;if(c==a[4]||c==a[5]){s=sigblock(
8192);printf("\033[H\033[J\033[0m%d\n",w);if(c==a[5])break;for(j=264;j--;Q[j]=
0);while(getchar()-a[4]);puts("\033[H\033[J\033[7m");sigsetmask(s);}}d=popen(
"stty -cbreak echo stop \023;sort -mnr -o HI - HI;cat HI","w");fprintf(d,
"%4d from level %1d by %s\n",w,l,getlogin());pclose(d);}
```

Slide 7



Slide 8

Warum nicht C++ für Betriebssysteme?

- C++ enthält alle Fallstricke von C plus weitere Fallstricke ⇒ C++ schwerer zu erlernen
- schwerer zu überschauen ⇒ durch Einsatz falscher Mittel werden die compilierten Programme oft dick und lahm
- C++ erfordert umfangreiches Laufzeitsystem, das ins Betriebssystem integriert werden müßte

Warum nicht Modula-2 oder Modula-3 für Betriebssysteme?

Slide 9

zwar leichter zu erlernen, Modulunterstützung und weniger Fallen, aber

- nicht so verbreitet
- nicht so portabel
- M-2 nicht ausdrucksstark genug

Geschichte der Programmiersprache C

Slide 10

1971–1973 C entsteht als Implementationssprache für UNIX als Nachfolger für B, dessen Vorgänger BCPL war.

1973–1982 C und UNIX tragen jeweils zum Erfolg des anderen bei und werden weltweit verwendet

1983 Kernighan/Ritchie veröffentlichen „The C Programming Language“, die Definition und Referenz für „K&R C“, dem klassischen Dialekt.

1983–1989 Standardisierungsbestrebungen führen zu „ANSI C“, einem modernisierten Dialekt

1990 ANSI C mit kleinen Veränderungen wird durch ISO zum internationalen Standard

Slide 11 $\Rightarrow$ 2 Dialekte. Wir beschäftigen uns ausschließlich mit ANSI C.

3 Überblick: Hello, World!

Slide 12

```
#include <stdio.h>              /* Präprozessoranweisung: lade
                                   Standard-Deklarationen
                                   für I/O */
int main(void)                  /* Beginn Definition Funktion
                                   main() */
{
  printf("Hello, World!\n");    /* rufe Funktion printf() mit
                                   Stringkonstante als Parameter
                                   auf */
  return 0;                     /* Liefere Ergebis 0 (= OK) an
                                   Vaterprozeß zurück */
}
```

Erkennbare Eigenschaften:

- Präprozessor kann Include-Dateien mit Deklarationen einbinden
- Jedes C-Programm enthält eine Funktion `main()`, die als Hauptprogramm beim Programmstart angesprungen wird.

  Ergebnis von `main()` wird an Vaterprozeß zurückgeliefert.

  („Funktion“ synonym für „Prozedur“)
- Zeichenkettenkonstanten werden in doppelte Hochkommas eingeschlossen und können Escape-Zeichen enthalten („`\n`“ = Newline o. Linefeed = LF = ASCII 0xa).
- `printf()` ist kein in C eingebauter „Befehl“, sondern eine Funktion in der Standard-Bibliothek, die zu allen Programmen hinzugebunden wird.
- `return` ist keine Funktion (kein Aufrufoperator „`()`“ notwendig), sondern eine C-Anweisung.
- Kommentare eingeschlossen in `/* ... */`.

Aufruf des GNU C-Compilers

Warum GNU C Compiler?

- hohe Verfügbarkeit
- unterstützt Anfänger durch viele Warnungen

Slide 13

Aufruf im Terminal, wenn obiges Programm in Datei `hello.c` vorliegt:

```
$ gcc -Wall -o hello hello.c
```

Aufruf des compilierten Programms:

```
$ ./hello
Hello, World!
$
```

Online-Hilfe unter Unix

Das Unix-Referenzmanual ist elektronisch online verfügbar.

Hilfe abrufbar z.B. mit

Slide 14

```
$ man gcc
$ man 3 printf
```

oder mit fensterorientiertem Programm XMan:

```
$ xman
```

4 Struktur eines C-Programms

Ein C-Programm enthält:

Slide 15

- Präprozessoranweisungen (z.B. `#include`)
- Typdefinitionen
- Funktionsdeklarationen (Prototypen): deklarieren, welche Parameter eine Funktion erwartet und welchen Typs ihr Ergebnis ist)
- Variablendeklarationen: deklarieren den Typ von Variablen
- Funktionsdefinitionen: lokale Deklarationen, Steueranweisungen und Ausdrucksberechnungen

Jedes C-Programm muß eine Funktion `main()` enthalten.

5 Datentypen

Variablen sind typisiert und müssen vor Gebrauch deklariert werden. I.A.:

```
int zahl;
```

Slide 16

5.1 Einfache Datentypen

Datentyp	**Typschlüsselwort**	**Bsp. f. Konstanten**

Zeichen	`char`
	`signed char`
	(`unsigned char`)
		`'a'`, `'\n'`, `15`

Slide 17

Datentyp	Typschlüsselwort	Bsp. f. Konstanten

Integer-Zahl	`int`
	`(signed int)`
	`unsigned int`
	`long int`
	`signed long int`
	`unsigned long int`
		`10`, `-15314`, `0731`, `0xdeadbeef`

Real-Zahl	`float` `double`	`1.3542e-23`, `-2345.2211`

Kein Boolean-Typ! Wahrheitswerte werden durch Integer-Variablen gespeichert: `0` = falsch, `!=0` = wahr.

5.2 Strukturierte Typen

Slide 18

Datentyp	Syntax	Beispiele f. Variablen-deklaration
Feld	*typ ID*[*dimension*]	`int feld[100]`

String	`char` *ID*[*dimension*]	`char string[100]`
		`"das ist ein String"`
		`"Zeile 1\nZeile 2"`

Zeiger	*typ* *	`int *p_int` `char *s`

Felder werden stets mit 0 beginnend indiziert.

String-Konstanten werden als `char`-Felder mit anhängter Null (0x0, `'\0'`) im Speicher abgelegt.

Felder und Zeiger eines Typs sind zuweisungskompatibel:

Slide 19

```
p_int = feld;                    /* p_int zeigt auf
                                    0. Element von feld */
s = "das ist ein String";        /* s zeigt auf 'd' */
s = string;
```

Beispiel

Slide 20

```
#include <stdio.h>

char name[] = "Zweites Programm.\n";
char *ausgabe;

int main(void)
{
  ausgabe = name;

  printf(ausgabe);

  return 0;
}
```

- Deklaration zweier globaler Variablen
- Feld `name` wird mit String initialisiert; da dem Compiler die Größe des Initializers (String-Konstante) bekannt ist, muß keine Felddimensionierung angegeben werden
- Zuweisung Feld zu Zeiger
- Aufruf `printf()` mit Zeiger-Argument

Slide 21

Datentyp	Syntax	Beispiele f. Variablen-deklaration
Aufzähl-Typ	`enum [ ID ] { ID [= val], ... }`	`enum { GELB = 0, GRUEN }`
Struktur	`struct [ ID ] { var.-dekl.; ... }`	`struct foo { int a; char *s; }`
Union	`union [ ID ] { var.-dekl.; ... }`	`union bar { int a; char *s; }`

5.3 Benutzerdefinierte Typen

`typedef` *`var-dekl`* `;`

Beispiele:

Slide 22

```
typedef int feldtyp[100];       /* Typdeklaration */
feldtyp feld;                   /* Variablendeklaration */

typedef char *string;
string name;
```

6 Variablendeklaration

globale Variablen: werden außerhalb von Funktionen deklariert

lokale Variablen: können zu Beginn eines jeden Blocks ($\{\ldots\}$) deklariert werden

Parametervariablen: aktuelle Parameter einer Funktion

Slide 23

Variablen kann durch eine Speicherklasse zugeordnet werden, die Gültigkeitsbereich und Lebensdauer beeinflussen:

Slide 24

<table>
<tr><th>Schlüsselwort</th><th>globale Variablen</th><th>lokale Variablen</th><th>Parametervariablen</th></tr>
<tr><td>(ohne)</td><td>programmglobale Gültigkeit; „common“</td><td>= <code>auto</code></td><td>lebt bis Verlassen der Funktion</td></tr>
<tr><td><code>auto</code></td><td>—</td><td>lebt bis Verlassen des Blocks</td><td>—</td></tr>
<tr><td><code>static</code></td><td>nur in diesem Modul gültig</td><td>Wert bleibt nach Verlassen bestehen</td><td>—</td></tr>
<tr><td><code>extern</code></td><td colspan="2">Variable in anderem Modul programm-global definiert</td><td>—</td></tr>
<tr><td><code>register</code></td><td>—</td><td colspan="2">wenn möglich, in Registern ablegen</td></tr>
</table>

Beispiel

```
static int wie_oft = 0;

void init(void)
{
  static int wurde_initialisiert = 0;

  if (! wurde_initialisiert)
    {
      /* führe Initialisierung durch... */
      wie_oft += 1;

      wurde_initialisiert = 1;
    }
}
```

Slide 25

7 Ausdrücke und Operatoren

Slide 26

Außer Deklarations- und Steueranweisungen enthalten C-Programme nur noch Ausdrucksberechnungen. Alle Ausdrucksanweisungen sind mit einem Semikolon abgeschlossen. Beispiele:

```
variable1;
a = b + c;
funktion();
```

Operatoren in der Reihenfolge ihrer Ausführung (Operatoren einer Gruppe haben die gleiche Priorität und werden von rechts nach links ausgeführt):

Slide 27

Operator	**Bedeutung**	**Beispiel**
Klammernoperator		
`( )`		`(a + b) * c`
Aufruf- und Selektionsoperatoren		
`()`	Funktionsaufruf	`sin(x)`
`[]`	Feldindizierung	`stunde[0]`
`.`	Strukturauswahl	`person.name`
`->`	Strukturauswahl (Zeiger) (`a->b` entspricht `(*a).b`)	`person->name`

Slide 28

Operator	**Bedeutung**	**Beispiel**
Unäre Operatoren		
`!`	logisches NICHT	`! initialized`
`~`	bitweises NICHT	`~0xffff`
`++`	Inkrementieren	`a++, ++b`
`--`	Dekrementieren	`a--, --b`
`+`	positives Vorzeichen	`+a`
`-`	negatives Vorzeichen	`-b`
`*`	Zeiger-Dereferenzierung	`*zeiger`
`&`	Adreßoperator	`&var`
`sizeof`	Größe	`sizeof int`
`(`*typ*`)`	Typumwandlung	`(long *) 0x1000`

Slide 29

Operator	**Bedeutung**	**Beispiel**
Binäre Punkt-Operatoren		
`*`	Multiplikation	`a * b`
`/`	Division	`a / b`
`%`	Divisionsrest (modulo)	`a % b`
Binäre Strich-Operatoren		
`+`	Addition	`a + b`
`-`	Subtraktion	`a - b`
Bitschiebeoperationen		
`<<`	bitweises Schieben links	`a << 2`
`>>`	bitweises Schieben rechts	`a >> 2`

Slide 30

Operator	**Bedeutung**	**Beispiel**
Relationale Operationen		
`<`	kleiner als	`a < b`
`>`	größer als	`a > b`
`<=`		
`>=`		
`==`	Gleichheit	`a == b`
`!=`	Ungleichheit	`a != b`
Bitoperationen		
`&`	bitweises UND	`a & b`
`^`	bitweises EXKLUSIV-ODER	`a ^ b`

Slide 31

<table>
<tr><th>Operator</th><th>Bedeutung</th><th>Beispiel</th></tr>
<tr><td><code>|</code></td><td>bitweises ODER</td><td><code>a | b</code></td></tr>
<tr><td colspan="3">Logische Operationen</td></tr>
<tr><td><code>&&</code></td><td>logisches UND</td><td><code>this && that</code></td></tr>
<tr><td><code>||</code></td><td>logisches ODER</td><td><code>this || that</code></td></tr>
<tr><td colspan="3">Bedingungsausdruck</td></tr>
<tr><td><code>? :</code></td><td></td><td><code>(a > b) ? a : b</code></td></tr>
<tr><td colspan="3">Zuweisungsoperatoren</td></tr>
<tr><td><code>=</code></td><td>einfache Zuweisung</td><td><code>a = b</code></td></tr>
<tr><td><code>+=</code></td><td><code>a += b</code> entspricht <code>a = a + b</code></td><td><code>a += b</code></td></tr>
<tr><td><code>-=</code></td><td></td><td></td></tr>
</table>

Slide 32

<table>
<tr><th>Operator</th><th>Bedeutung</th><th>Beispiel</th></tr>
<tr><td><code>*=</code></td><td></td><td></td></tr>
<tr><td><code>/=</code></td><td></td><td></td></tr>
<tr><td><code>%=</code></td><td></td><td></td></tr>
<tr><td><code>&=</code></td><td></td><td></td></tr>
<tr><td><code>^=</code></td><td></td><td></td></tr>
<tr><td><code>|=</code></td><td></td><td></td></tr>
<tr><td><code><<=</code></td><td></td><td></td></tr>
<tr><td><code>>>=</code></td><td></td><td></td></tr>
<tr><td colspan="3">Kommaoperator</td></tr>
<tr><td><code>,</code></td><td>erst a, dann b</td><td><code>a, b</code></td></tr>
</table>

Operatoren in Variablen-/Typdeklarationen

Es gelten die gleichen Vorrangregeln:

```
int (*table[10])(double);
```

`table` ist ein 10-elementiges Feld von Zeigern auf Funktionen, die ein `double`-Argument erwarten und `int` zurückliefern.

Slide 33

```
int (*f)(double)[10];
```

`f` ist ein Zeiger auf eine Funktion, die ein `double`-Argument erwartet und einen Zeiger auf ein 10-elementiges `int`-Feld zurückliefert. Da Feldgrenzen nicht überprüft werden, ist folgende Deklaration identisch:

```
int * (*f)(double);
```

⇒ „von innen nach außen“

8 Anweisungen

8.1 Leere Anweisung

Slide 34

;

8.2 Einfache Anweisung

ausdruck ;