Die Programmiersprache C

1. Überblick und Einführung

Vergleich, Motivation, Historie, Technologie

Prof. Dr. sc. Joachim Fischer (Dr. Klaus Ahrens)
Institut für Informatik, Humboldt-Universität zu Berlin
SS 2014

Java versus C

Java

C

Java versus C

- **Java** ist ein experimentelles Fahrzeug auf Luftkissenbasis. Es bewegt sich auf Straßen aller Art, ist allerdings noch schwer zu steuern. Es ist strengstens verboten, Umbauten am Fahrzeug vorzunehmen.

- **C** ist ein offener Geländewagen. Kommt durch jeden Matsch und Schlamm, aber der Fahrer sieht hinterher auch dementsprechend aus.

Eigenschaften von C

Einige Eigenschaften

- kompakte Sprache – Reduktion auf das “Wesentliche”
- extensive Nutzung von Prozeduraufrufen/Funktionen
- schwaches Typ-Konzept (im Gegensatz zu Java, PASCAL)
- aber: strukturierte Programmiersprache
- *low-level*: Bit-orientierte Programmierung ist möglich
- Zeiger (engl. Pointer) Implementation: extensive Nutzung von Zeigern für Speicher, Arrays, Strukturen und Funktionen (später)
- effizienter Code kann erzeugt werden
- Portabilität (durch Sprachstandard): es kann Code für verschiedene Rechner erzeugt werden, aber breite „Grauzone“

Die Stärken von Java

- einfach
- objektorientiert
  Simula-67(1970), Smalltalk(1975), C++(1987), Eiffel(1988)
- architekturunabhängig, interpretativ
  virtuelle Maschine mit Bytecode als Interpreter: P-Code (Pascal, 1974),
  S-Code(Simula, 1978), M-Code(Modula, 1982)
- getrennte Übersetzung
  Modula-2 (1982)
  C, C++ nur textuelle Inklusion von Header-Files
- typsicher (zuweisungs- und ausdruckskompatibel)
  aber keine Pointer und Adressen (Maschinenorientierung)
  Indextest bei Feldern, "Garbage Collection" von Daten-Objekten

Schwächen von C

- aus softwaretechnischer und sprachtheoretischer Sicht ein gewisser Rückschritt (1978)
- einige unsaubere (unregelmäßige) Sprachkonzepte 
- unsichere Sprache (es wird weniger geprüft als in Java), nur unabhängige statt separate Compilation (damit fehleranfälliger) 
- Beispiel: **keine** implizite Initialisierung lokaler Variablen 

Stärken von C

- flexibel
- Präprozessor (Makros, Include-Files, bedingte Compilation)
- Maschinennähe (*low-level*-Konzepte), damit hohe Laufzeiteffizienz erreichbar
- Betriebssysteme sind (häufig) in C programmiert
- Objektorientierte Erweiterungen existieren (mit Beibehaltung der Stärkung und Verringerung der Schwächen von C)
- *Zero-Overhead*-Prinzip: „Sie zahlen nicht für Dienstleistungen, die Sie nicht in Anspruch nehmen“ ☺

Ein praktischer Vergleich

- Wie gut eignen sich beide Sprachen „große“ Probleme „schnell“ zu lösen?
- Musterszenario: „Viele“ Zeichenketten (im Speicher) lexikographisch sortieren.
- Hier: zufällig erzeugte Strings (aus kleinen Buchstaben) werden (*in situ* mittels Quicksort) sortiert.
- Aufruf:
  - **`[time] java s N`**
  - **`[time] s N`** - N = Anzahl der Strings

Eine Lösung in Java

```
public class s {
  static java.util.Random rand = new java.util.Random();

  static public void main(String[]s)
  {
      if (s.length < 1) return;
      int N = Integer.parseInt(s[0]);

      String[] all = new String[N];

      fill (all, N);

      java.util.Arrays.sort (all);
      //out (all, N);
  }
```

Eine Lösung in Java

```
static private void fill (String[] v, int n) {
    for (int i=0; i<n; ++i)
          v[i] = randomString();
}

static private String randomString() {
    char[] chars = new char [10];
    for (int i=0; i<length; ++i)
          chars [i] = (char)('a'+ rand.nextInt(26));
    return new String (chars);
  }

  static private void out(String[] v, int n) {
    for (int i=0; i<n; ++i)
          System.out.println(v[i]);
  }
}
```

Eine Lösung in C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LENGTH 10

void  fill(char**, int);
char* randomString();
void  out(char**, int);
int   compare(const void*, const void*);

int main(int argc, char** argv) {
   int N;
   char** all;
   if (argc < 2) return 0;

   N = atoi(argv[1]);
   all = (char**)malloc(N * sizeof(char*));

   fill (all, N);

   qsort (all, N, sizeof(char*), compare);
   /* out (all, N); */
   return 0;
}
```

Eine Lösung in C

```
void fill (char** v, int n) {
   int i;
   for (i=0; i<n; ++i)
        v[i]=randomString();
}

char* randomString() {
   char* chars = (char*)malloc(LENGTH + 1);
   int i;
   for (i=0; i<LENGTH; ++i)
        chars[i] = ('a' + rand() % 26);
   chars[LENGTH] = 0;

   return chars;
}

void out(char** v, int n) {
   int i;
   for (i=0; i<n; ++i)
        printf("%p: %s\n", v[i], v[i]);
}

int compare(const void* x, const void* y) {
   return strcmp(*(char**)x, *(char**)y);
}
```

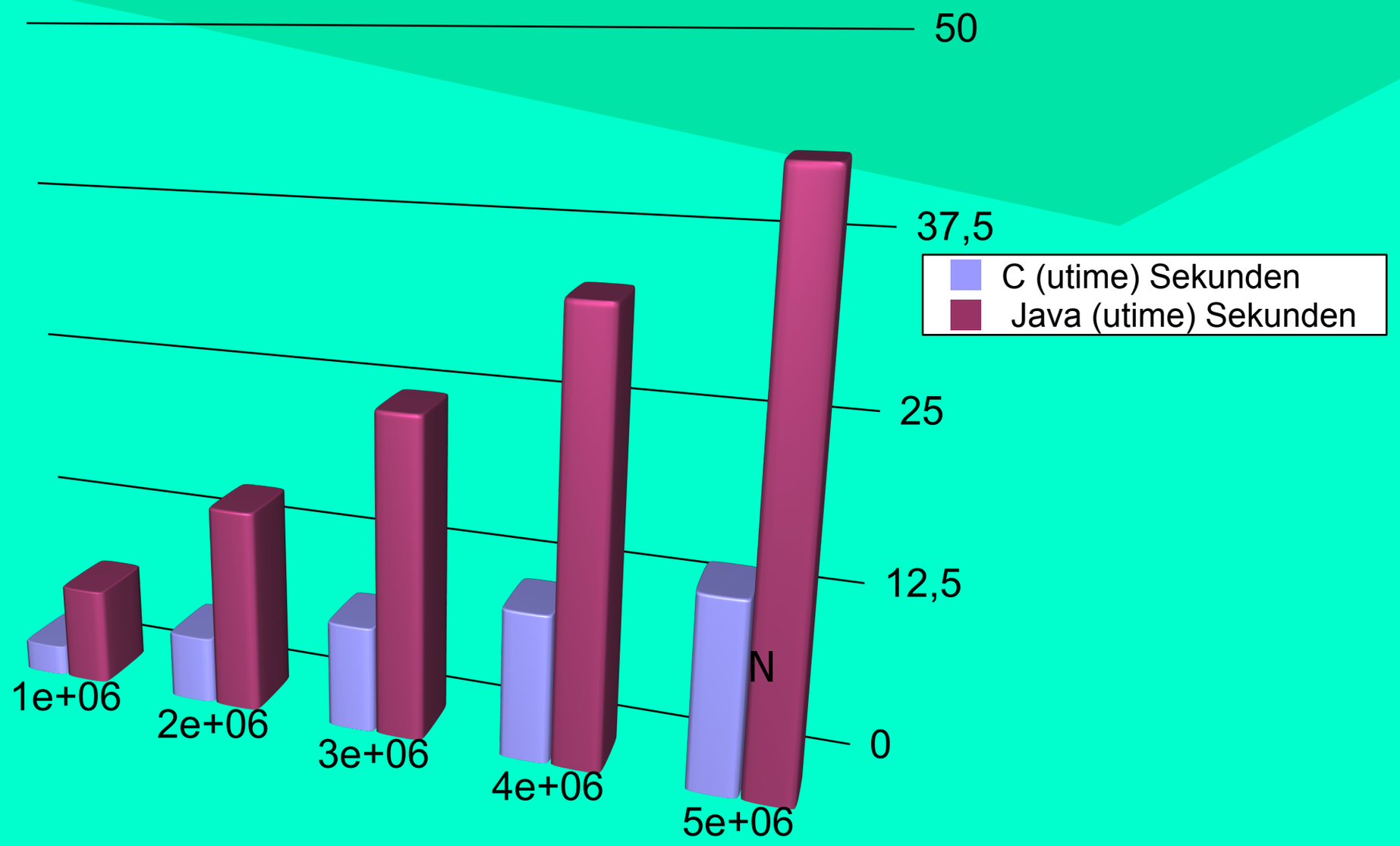
Vergleich

- Starke strukturelle Ähnlichkeit

- Unterschiede in:
  - Einbettung in die (Bibliotheks-) Umgebung
  - Behandlung von Programmparametern
  - Funktionskontext (Klassen-lokal vs. global)
  - Notwendigkeit von Prototypen (Vorab-Deklarationen)
  - Typ von Zeichenketten
  - Speicherverwaltung

- Aber vor allem in den Laufzeiteigenschaften !!!

Vergleich der Laufzeiten



Geschichte der Sprache C

Markante Ereignisse

- UNIX wurde etwa 1969 entwickelt (auf einer DEC PDP-7 in Assembler)
- BCPL – eine Sprache mit mächtigen Entwicklungswerkzeugen
  - Erfahrung: Assembler als Entwicklungssprache zu “langatmig” und fehleranfällig
- Sprache “B” als weiterer Anlauf um 1970
- dritter Anlauf: neue Sprache mit neuen Ansätzen (geprägt durch Pascal/Algol): C als Nachfolger von B (um 1971)
- bis 1973 wurde UNIX fast vollständig in C umgeschrieben
- Ur-C: Kernighan & Ritchie
- heute aktuell ISO(ANSI)-C==C89, C99, C11

Geschichte der Sprache C

COMPUTERWORLD 1 April 1991
CREATORS ADMIT UNIX, C HOAX

In an announcement that has stunned the computer industry, Ken Thompson, Dennis Ritchie and Brian Kernighan admitted that the Unix operating system and C programming language created by them is an elaborate April Fools prank kept alive for over 20 years. Speaking at the recent UnixWorld Software Development Forum, Thompson revealed the following:

"In 1969, AT&T had just terminated their work with the GE/Honeywell/AT&T Multics project. Brian and I had just started working with an early release of Pascal from Professor Nichlaus Wirth's ETH labs in Switzerland and we were impressed with its elegant simplicity and power. Dennis had just finished reading 'Bored of the Rings', a hilarious National Lampoon parody of the great Tolkien 'Lord of the Rings' trilogy. As a lark, we decided to do parodies of the Multics environment and Pascal. Dennis and I were responsible for the operating environment. We looked at Multics and designed the new system to be as complex and cryptic as possible to maximize casual users' frustration levels, calling it Unix as a parody of Multics, as well as other more risque allusions. Then Dennis and Brian worked on a truly warped version of Pascal, called 'A'. When we found others were actually trying to create real programs with A, we quickly added additional cryptic features and evolved into B, BCPL and finally C. **We stopped when we got a clean compile on the following syntax:**

```
for(;P("\n"),R-;P("|"))for(e=C;e-;P("_"+(*u++/8)%2))P("|  "+(*u/4)%2);
```

...

Web-Ressourcen

Können wie üblich wandern / verschwinden ☺

Eine Sammlung nützlicher Fragen mit Antworten:

http://www.eskimo.com/~scs/C-faq/top.html

Wertvolle Hinweise zum Programmierstil:

http://www.jetcafe.org/~jim/c-style.html

Eine (inoffizielle, weil Vorab-) Version des C(99)-Standards (in einem File):

http://busybox.net/~landley/c99-draft.html

Eine Sammlung von weiteren Links:

http://www.lysator.liu.se/c/

Coding Style

- In C lassen sich viel einfacher schlechte (unübersichtliche, schwer verständliche, schwer wartbare, nicht portable, ...)

Programme schreiben ☹

- `ioccc.org` – The International Obfuscated C Code Contest

```
float o=0.075,h=1.5,T,r,O,l,I;int _,L=80,s=3200;main(){for(;s%L||
(h-=o,T= -2),s;4 -(r=O*O)<(l=I*I)|++ _==L&&write(1,(--s%L?_<L?--_
%6:6:7)+"World! \n",1)&&(O=I=l=_=r=0,T+=o /2))O=I*2*O+h,I=l+T-r;}
```

Coding Style

```
#include <stdio.h>
#include <math.h>
#define E return
#define S for
char*J="LJFFF%7544x^H^XXHZZXHZ]]2#( #@@DA#(.@@%(0CAaIqDCI$IDEH%P@T@qL%PEaIpBJCA\
I%KBPBEP%CBPEaIqBAI%CAaIqBqDAI%U@PE%AAaIqBcDAI%ACaIaCqDCI%(aHCcIpBBH%E@aIqBAI%A\
AaIqB%AAaIqBEH%AAPBaIqB%PCDHxL%H@hIcBBI%E@qJBH#C@@D%aIBI@D%E@QB2P#E@'C@qJBHqJBH\
%C@qJBH%AAaIqBAI%C@cJ%"   "cJ"   "CH%C@qJ%aIqB1I%PCDI`I%BAaICH%KH+@'JH+@KP*@%S@\
3P%H@ABhIaBBI%P@S@PC#",     *j   ,*e;typedef float x;x U(x a){E a<0?0:a>1?1:a; }
typedef struct{x c,a,t;      }   y;y W={1,1,1},Z={0,0,0},B[99],P,C,M,N,K,p,s,d,h
;y G(x t,x a,x c){K.c=t   ;      K.t=c; K.a=a;E K;}int T=-1,b=0,r,F=-111,(*m)(i\
nt)=putchar,X=40,z=5,o,   a,     c,t=0   ,n,R;y A(y a,y b,x c){E G(a.c+b.c*c,a.a
+c*b.a,b.t*c+a.t);}x H=   .5,Y   =.66     ,I,l=0,q,w,u,i,g;x O(y a,y b){E q=a.t*
b.t+b.c*a.c+a.a*b.a;}x Q(){E A(P,M,T      ),O(K,K)<I?C=M,I=q:0;}y V(y a){E A(Z,
a,pow(O(a,a),-H));}x D(y p){S(I=X,P       =p,b=T; M=B[++b],p=B[M.c+=8.8-l*.45,
++b],b<=r;Q())M=p.t?q =M_PI*H,w=atan2(   P.a-M.a,P.c-M.c)  /q,o=p.c-2,a=p.a+1,t=
o+a,w=q*(w>t+H*a?o:     w>t?t:w<o-H*a?t   :w<o?o:w),A(      M,G(cos(w),sin(w),0),
1):A(M,p,U(O(A(P,M,T)     ,p)/O(p,p)));   M=P;M.a=-         .9;o=P.c/8+8;o^=a=P.t
/8+8; M=Q        ()?o&1    ?G(Y,0,0):W    :G(Y,Y,1);E       sqrt        (I)-.45;}
int main(   int    L,char    **k){ S(e    =L>1?1[z=    0,   k]:J   ;*e   &&l<24 ;
++e)S(o=a   =0,j   =J+9;(c=    *++j)&&   !(o&&c<    X&&(q=l+=w)   );o   ?o=*j++/
32,b++[B]   =G(q   +=*j/8&3,*    j&7,0    ),B[r    =b++]=G((c/8&       3)*( o<2?
T:1), (c&   7)+    1e-4,o>2),1:    (o     =(a    =(c-=X)<0?w=c+6   ,t=   a+1:c?(t
?0:m(c),a        ):*++j)==((*e|32          )    ^z)&&1[j]-X));S(z   =3*(   L<3);++
F<110;)S(L=-301;p=Z,++L<300;m(              p.c),m(p.a),m(p.t))S(c=T;++c<=z;)S(h
=G(-4,4.6,29),d=V(A(A(A(Z,V(G(5,0           ,2)),L+L+c/2),V(G(2,-73,0)),F+F+c%2),G
(30.75,-6,-75),20)),g=R=255-(n=z)*64;    R*n+R;g*=H){S(u=i=R=0;!R&&94>(u+=i=D(h=
A(h,d,i)));R=i<.01);S(N=V(A(P,C,                 T)),q=d.t*d.t,s=M,u=1;++i<6*R;u-=
U(i/3-D(A(h,N,i/3)))/pow(                          2,i));s=R?i=pow(U(O(N,V(A(
M=V(G(T,1,2)),d,T))))                                  ,X),p=A(p,W,g*i),u*=U(
O(N,M))*H*Y+Y,g*=                                        n--?Y-Y*i:1-i,s:G(
q,q,1); p=A(p,s                                            ,g*u);h=A(h,N,.1
);d=A(d,N,-2*O                                               (d,N));}E 0;}
```

IOCCC 2011 winner

Matt Zucker

http://www.ioccc.org/2011/zucker/hint.html

```
$ ./zucker > image.ppm
```

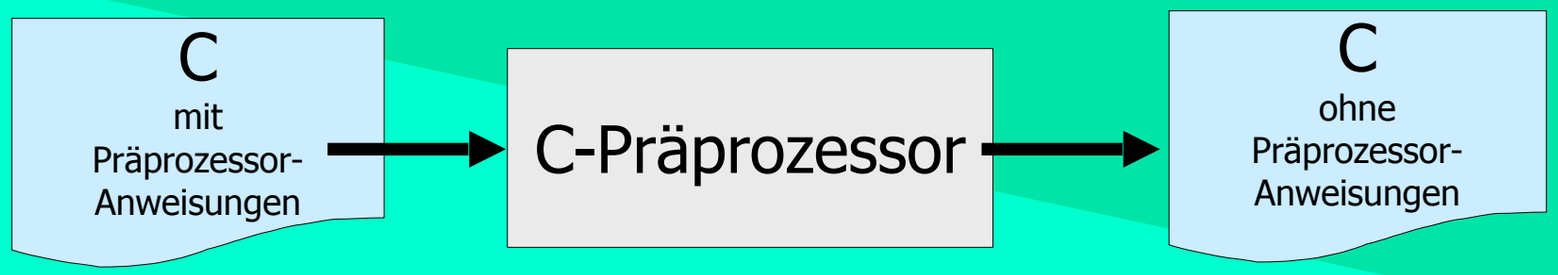
Coding Style

Gerade deshalb ist es umso wichtiger, gute (übersichtliche, leicht verständliche, gut wartbare, portable, ...) Programme zu schreiben

Wie?

- Übertragen Sie ihren Java-Style auf C!
- Divide et impera! (kleine Funktionen, kleine Quelltexte)
- Formatieren/strukturieren/kommentieren Sie lesbar!
- Nehmen Sie Warnungen ernst!
- Vermeiden Sie Hacks!
- Nutzen Sie verschiedene Compiler auf verschiedenen Plattformen
- Nutzen Sie Test- und Analysewerkzeuge (z.B. **`valgrind`**)
- Testen Sie abschließend (und nicht zu spät) auf der Referenzplattform

Compilertechnologie



- Einfügen von Dateien

```
#include <stdio.h>
#include "stack.h"
```

(Systemdatei)
(nutzerdefiniert)

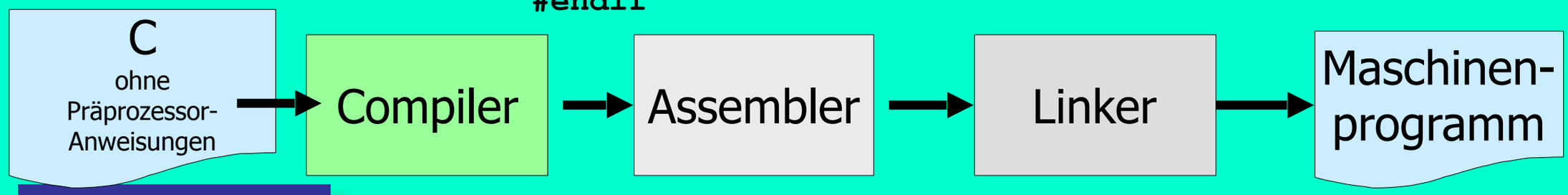
- Ersetzen von Text(Makros)

```
#define L 5
#define add100(x)  ((x)+100)
```

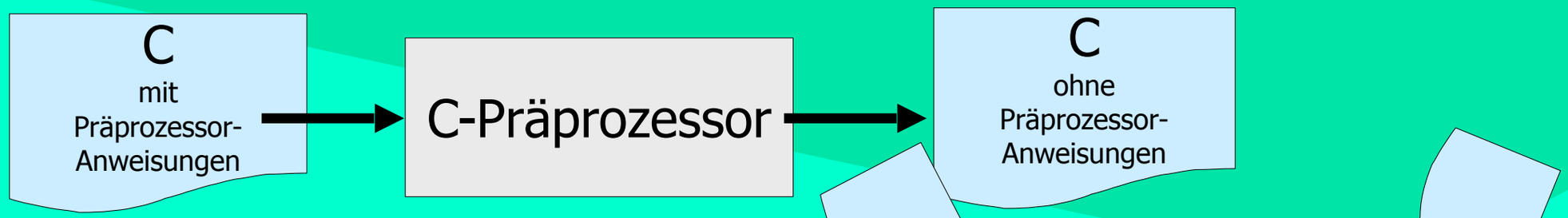
(Konstantendefinition)
(parametrisiert)

- bedingte Übersetzung

```
#ifdef TEST
                printf("Testversion");
#else
                printf("Produktionsversion");
#endif
```



Compilertechnologie



- Einfügen von Dateien

```
#include <st                    (Systemdate
#inc de "stac                  (nutzerdefin
```

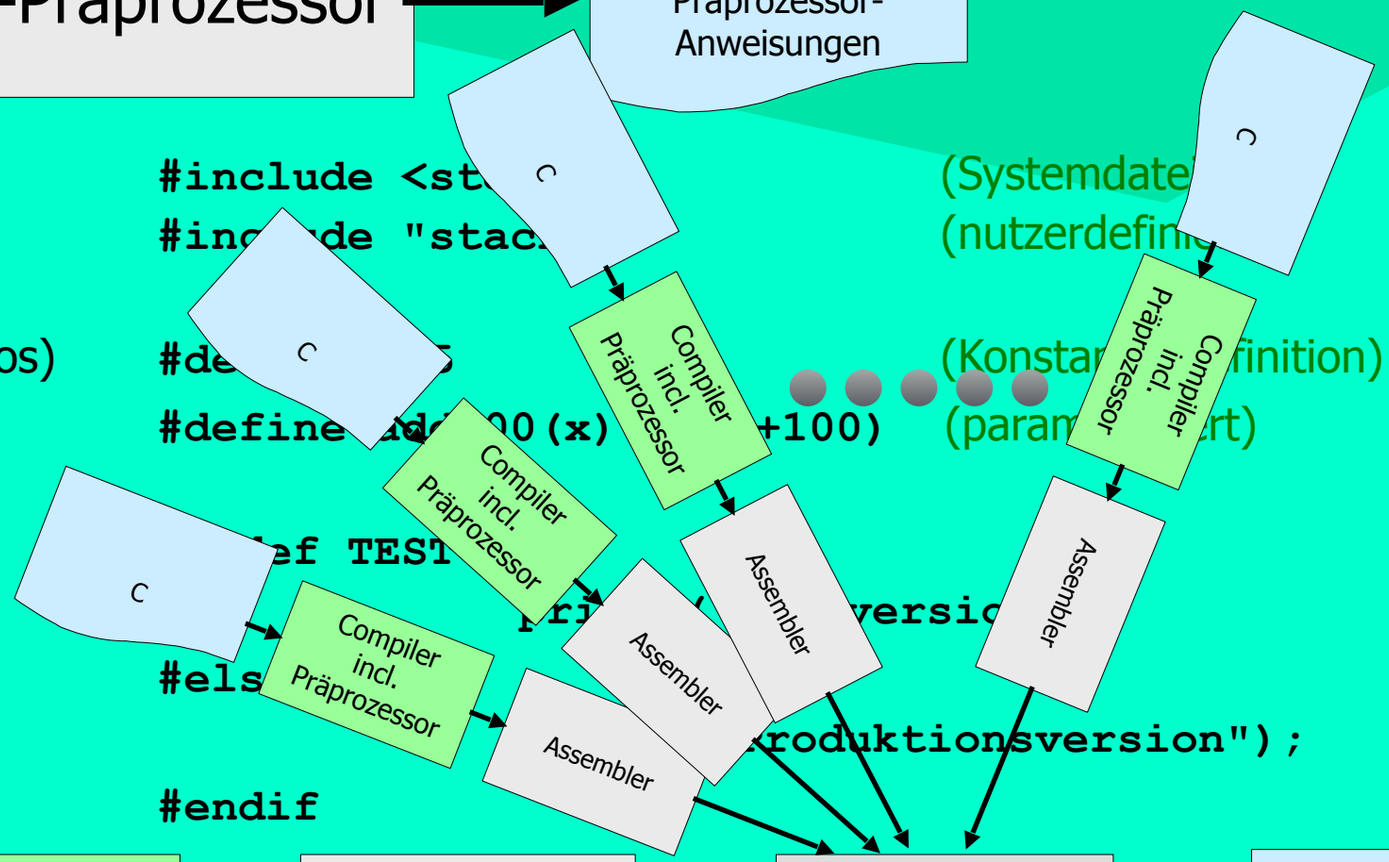
- Ersetzen von Text(Makros)

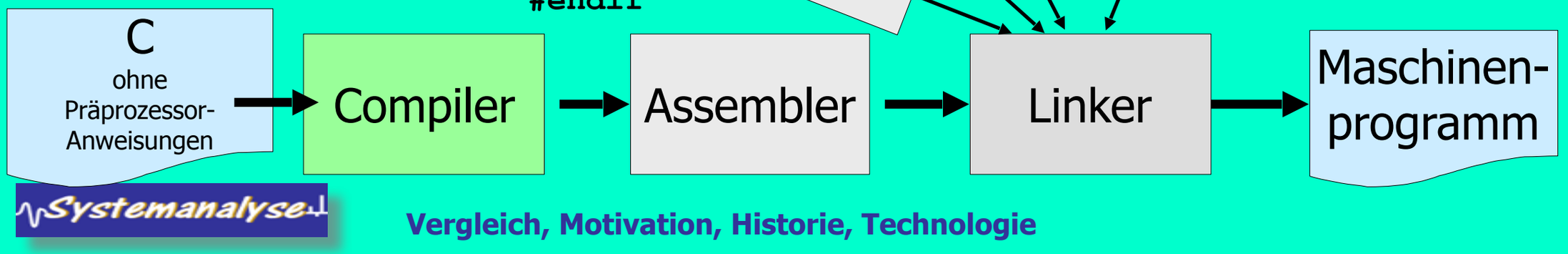
```
#de                            (Konstan finition)
#define  00(x)      +100)      (param rt)
```

- bedingte Übersetzung

```
   ef TEST
   ri                  versio
#els                   roduktionsversion");

#endif
```





Compilertechnologie

- unser Referenzsystem:

```
$ hostname
star
$ uname -a
SunOS star 5.11 11.1 sun4u sparc SUNW,SPARC-Enterprise
$ gcc --version
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- andere Compiler können benutzt werden, machen Sie **immer** auch Tests mit dem Referenzsystem (nicht erst kurz vor Abgabe)

- **`gcc`** steuert **ALLE** Phasen der Übersetzung: kann beliebige Zwischencodes der Übersetzung erzeugen **UND** weiterverarbeiten

`gcc -c x.c`	- *compile only*, erzeugt (falls fehlerfrei) **`x.o`**
`gcc -E x.c`	- *preprocess only*, Ausgabe nach stdout
`gcc -S x.c`	- *generate asm*, erzeugt (falls fehlerfrei) **`x.s`**
`gcc m.c`	- *compile and link*, erzeugt (falls fehlerfrei) ausführbares *binary* **`a.out`**
`gcc -o prog m.c`	- *compile and link*, erzeugt (falls fehlerfrei) ausführbares *binary* **`prog`**
`gcc -c x.c y.c z.c`	- *compile only*, erzeugt (falls fehlerfrei) **`x.o, y.o, z.o`**
`gcc -o p x.o y.o z.o`	- *link only*, erzeugt (falls fehlerfrei) ausführbares *binary* **`p`**
`gcc -o p x.s y.o z.c`	- *compile and link*, erzeugt (falls fehlerfrei) ausführbares *binary* **`p`**

Compilertechnologie

weitere Optionen

`-Wall`	- *warn all*, erzeugt (alle) Warnungen
`-Wall -pedantic`	- *warn all, pedantic* erzeugt noch mehr Warnungen
`-g`	- *instrument for debu**g**ging*
`-pg`	- *instrument for **p**rofilin**g***
`-Ox [x=1,2,3,s]`	- *optimize(1) more (2) and yet more (3)* for speed, *optimize(s)* for size
`-ansi`	- *accept ANSI-C* (C89==C90)
`-std=cxx [xx=89,90,99]`	- *accept ANSI-C* (C89==C90), *accept C99*
`-Dmacro[=defn]`	- *define macro* (as if **`#define macro defn`** in source file)
`-Umacro`	- *undefine macro* (as if **`#undef macro`** in source file)
`-Ipath`	- search for headers in *path* also
`-lxyz`	- link with *libxyz.a* (z.B. **`-lm`** manchmal nötig für **`libm.a`**)

und **Unmengen** weitere (**`man gcc, info gcc`**)

Die Programmiersprache C

2. Überblick und Einführung

Typen, Variablen, Funktionen, Operatoren

Prof. Dr. sc. Joachim Fischer (Dr. Klaus Ahrens)
Institut für Informatik, Humboldt-Universität zu Berlin
SS 2014

Sprachelemente

- ein C-Programm hat i. allg. folgenden Aufbau:
  - Präprozessor-Kommando(s)
  - Typdefinition(en)
  - Funktionsdeklaration(en):
    Deklaration von Funktionen mit Ein-und Ausgabeparameter(n) (ohne Definition)
  - Variablendefinition(en) und –deklaration(en)
  - Funktionsdefinition(en)

- Quelltext frei formatiert, Bezeichner wie üblich **`letter {letter | digit}*`** (_ is a letter, don't use in front)
- Kommentare eigentlich klassisch `/* … */` aber inzwischen auch fast überall `// …`
- Zeilenden können mit \ maskiert werden

```
"eine ziemlich lange \
zeichenkette"
```

- jedes ausführbare Programm besitzt genau eine (globale) **`main()`**-Funktion

```
int main() { … }                                    //  oder
int main(int someName1, char** someName2)  { … }    //  oder
int main(int someName1, char* someName2[]) { … }
```

Keywords

Reservierte Bezeichner in C (89, 99):

```
auto break case char complex const
continue default do double else enum
extern float for goto if imaginary inline
int long register restrict return short
signed sizeof static struct switch typedef
union unsigned void volatile while
```

Typen

Einfache (*built-in*) Typen

- fehlt: byte
- vorhanden: **`bool`** **`true/false`**

nicht in C89, in C99 via **`<stdbool.h>`**

`short, int, long`	ganze Zahlen unterschiedlicher Länge
`float, double`	reelle Zahlen unterschiedlicher (endlicher!) Genauigkeit
`char`	Zeichen: ASCII-Code (1 Byte)
(**`void`**)	kein wirklicher Typ: Funktionen ohne Resultat, **`void*`**

Zeigertypen (auf beliebige Typen)

Indirektion per Adresse

strukturierte (nutzerdefinierte) Typen

- Aufzählungen
- Felder (Arrays)
- Strukturen, Unions

Einfache Typen

C definiert die folgenden einfachen Typen: [mit **typischen** Implementationsgrößen]

C-Typ	**Größe (Byte)**	**Min-Wert**	**Max-Wert**
bool (C99)	1		
char	1	Plattform- abhängig :-(ASCII nutzt aber nur 7 Bit :-)	
unsigned char	1	0	255
short int	2	-32768	+32767
unsigned short int	2	0	65535
(long) int	4	-2^{31}	2^{31}
float	4	-3.4 x 10 [~ 6 Stellen genau]	+3.4 x 10
double	8	-1.7 x 10 [~ 15 Stellen genau]	+1.7 x 10
long long	8	-2^{63}	2^{63}

auch signed, aber
VORSICHT: Vorzeichen-behandlung u.U. nicht portabel

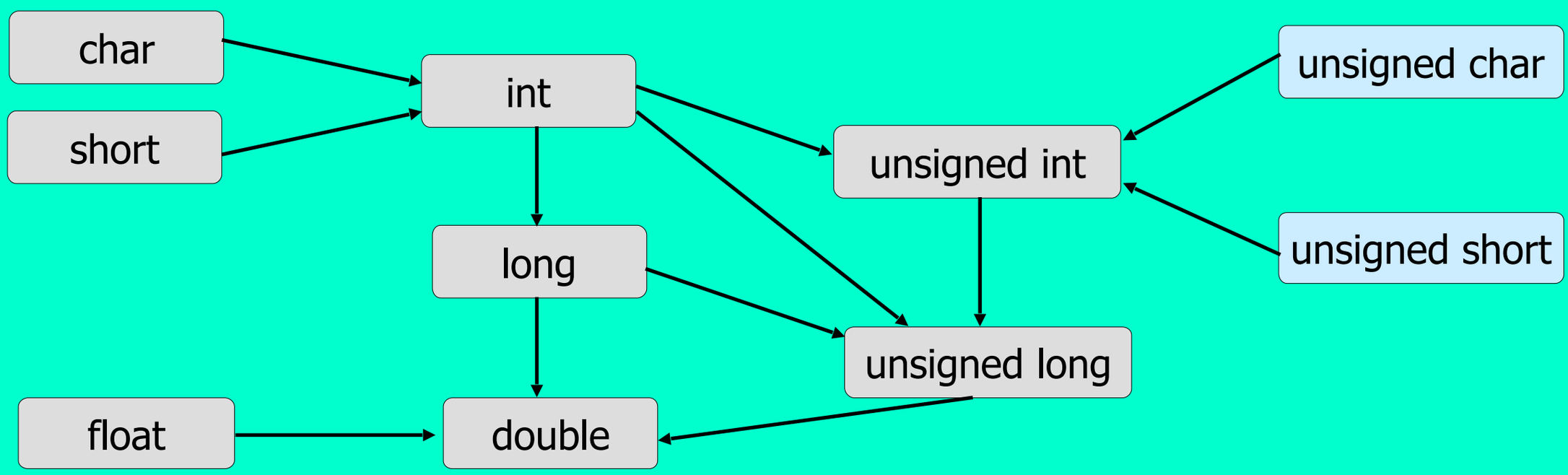
i.allg. zu ungenau
besser immer double

Typumwandlung - implizit

automatische Typanpassung in Ausdrücken oder Zuweisung **`dies = das;`**

z.B. 2-stelliger Operator:

Operanden werden immer in Richtung des umfassenderen Typs konvertiert



Typumwandlung - explizit

- durch Cast-Operationen

```
dies = (TypVonDies)das;
```

nur zwischen verwandten Typen erlaubt:

- arithmetische Typen aus der vorigen Folie (u.U. nicht portabel) 
- Zeigertypen (u.U. unsicher) 

Aufzählungstypen

- Aufzählungstypen setzen sich aus einer Liste von Konstanten zusammen, die auch als Integer-Werte benutzt werden können

  Beispiel:

```
enum days {mon, tues, ..., sun} day;
enum days day1, day2;
```

- wie bei Arrays hat der erste Name den Indexwert 0
  - mon hat Wert 0, tues den Wert 1, usw.
  - **`day1`** und **`day1`** sind Variablen
  - die Literale gelten im umgebenden Gültigkeitsbereich !
  - **`days`** ist **KEIN** eigentlicher Typname (nur *type tag*): **`typedef enum days Days;`**

- auch andere Werte sind möglich:

```
enum escapes { bell = '\a', backspace = '\b', tab = '\t',
               newline = '\n', vtab = '\v', return = '\r'};
```

- Anfangswert für Index kann auch überschrieben werden:

```
enum months {jan = 1, feb, mar, ......, dec}; // feb==2, mar ==3, …
```

Zeiger

Zeiger oder Pointer

- Das Zeigerkonzept ist ein wesentlicher Teil der Sprache C

- In C werden **Zeiger** intensiv genutzt. **Warum?**
  1. manchmal die einzige Möglichkeit (z.B. dynamische Variablen, siehe malloc/free)
  2. erzeugt kompakten und effizienten Code
  3. erlaubt direkten Zugang zum Hauptspeicher

- C nutzt Zeiger in Zusammenhang mit Feldern, Strukturen, Funktionen

Zeiger

- Ein Zeiger ist eine Variable, die eine Adresse einer anderen Variablen (als Wert) speichert.

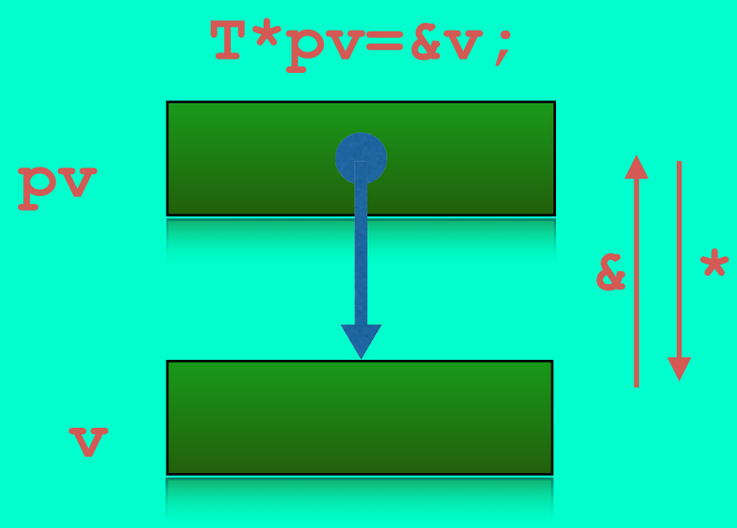
  Zeiger gibt es in C für Variablen beliebigen Typs

Beispiel für die Deklaration eines Zeigers:

```
int *pointer; int*p1; int * p2; int *p3;
int* p1, p2, p3; /* ACHTUNG: nur ein Zeiger, zwei integer */
```

- der monadische Adressoperator „**&**“ gibt die Adresse einer Variablen zurück:
  `T v;` —> `&v` hat den Typ `T*`

- Der Dereferenzierungsoperator „*“ gibt den „Wert eines Objektes zurück“, auf den der Zeiger zeigt



Zeiger

- ein Zeiger muss mit einem bestimmten Typ assoziiert werden
- nur Zeiger gleichen Typs sind zuweisungskompatibel
- es gibt allerdings einen universellen Zeigertyp **`void*`**, der beliebige andere Zeiger aufnehmen kann:

```
int* pi;                        double* px;
void* pv = pi;                  void* pv = px;

int* pj = (int*)pv;             double* py = (double*)pv;
```

```
py = (double*)pj;
// wird übersetzt, aber undefined behaviour
```